Chapter 6

Planning Systems

Chapter Contents

In the second chapter of this part we studied how fuzzy or expert systems could be used to represent human knowledge about how to perform control tasks. Essentially, they represent control software by emulating cognitive functionalities. Here, we focus on how to emulate the "software" functionality of more sophisticated reasoning strategies that use planning in order to decide how to control a plant. Since planning requires an ability to form representations (models) in the brain, exercise these representations to generate predictions about how the environment will react to various plans, choose among alternative plans, and execute a sequence of actions, it is only found in higher organisms (e.g., humans). While it certainly requires a neural network for implementation we do not focus on that; our focus here is on the functionalities basic to planning systems, and in particular, planning capabilities of humans as understood by psychologists.

Why is planning useful for control? Essentially, it is one approach that allows for more than simple reactions to what is sensed. It utilizes information about the problem and environment, often in the form of some type of model, and considers many options and chooses the best one to achieve the closed-loop control objectives. Planning provides for a very general and broadly applicable methodology and it has been exploited extensively in conventional control (e.g., in receding horizon control and model predictive control). As compared to the fuzzy and expert system approaches, it exploits the use of an explicit model to help it decide what actions to take. Like the fuzzy and expert system approaches, it is still, however, possible to incorporate heuristics that help to specify what control actions are the best to use. Hence, in a broad sense, planning approaches attempt to use both heuristic knowledge and model-based knowledge to make control decisions; this may be the fundamental reason for selecting a planning strategy over a simple rule-based one. It is often bad engineering practice to only favor the use of heuristics and ignore the information provided by a good mathematical model; planning strategies provide a way to incorporate this information.

6.1 Psychology of Planning

At an intuitive level, via introspection, you already understand what planning is. We plan our activities for the weekend, we plan a shopping trip, or plan how to solve a problem. A plan is a sequence of steps to achieve a goal, perhaps by performing tasks to achieve subgoals that then lead to the achievement of an overall goal.

Plans are typically hierarchical in that each task in sequence can often be viewed as a goal with a sequence of tasks to achieve it.

6.1.1 Essential Features of Planning

We often form "action plans" to try to achieve specific goals. For instance, consider Figure 6.1 where an "action hierarchy" is given as one type of action plan. Here, at the highest level there is the goal "eat lunch." Suppose that the person who is hungry, a professor who just got a job teaching at a university, develops a plan for how to achieve the goal to eat as the lunch hour approaches. The goal motivates the professor to pay *attention* to his hunger and pay attention to, and construct, a plan to meet his goal. This plan is formed using his past knowledge of what was successful for him when he was in graduate school, but modified somewhat due to his new role as a faculty member in a different university. In order to achieve his goal of eating lunch, he decides that he should consult the telephone directory given to him when he arrived since this may give him an idea of what restaurants are nearby. Next, since he is not familiar with any of the restaurants he found in the phone book, he asks some students and colleagues which restaurants have good food, are inexpensive, and yet have fast service. Notice then that some blocks in Figure 6.1 can be thought of as goals and tasks. Also, some of the blocks may need to be broken down further into tasks and goals. Next, the professor must pick a restaurant (based on personal tastes and priorities), find directions, choose a mode of transportation and route, and then travel to the restaurant. Hence, while a plan hierarchy may be conceptual, and as it is executed you may abstractly traverse the hierarchy, it may be that a subplan involves executing movements over a route that itself may be thought of as a planned path (e.g., the route to the restaurant). Clearly, there also may be a need for replanning, for example, if the planned route is unexpectedly blocked, or if the initial plan was in error due to someone providing bad directions. After arriving at the restaurant, the professor may execute a standard plan (a "script" available from his experience of eating at restaurants before) where he orders, eats, pays, and then returns to his office.

Learning and use of models for prediction is central to the activity of planning.

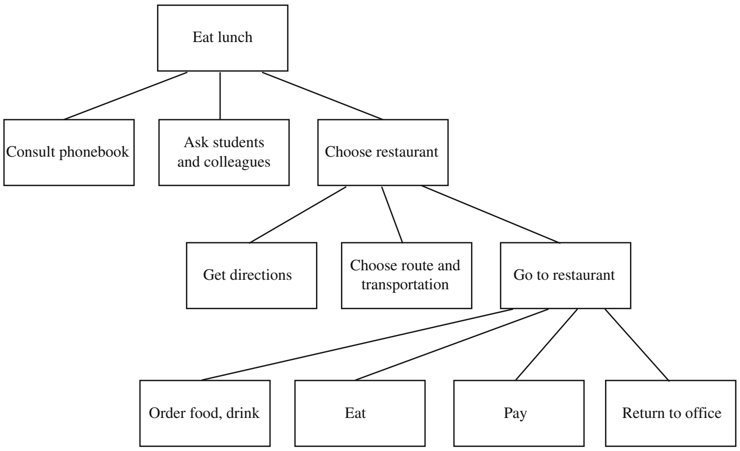

Figure 6.1: Action plan as an action hierarchy, an example.

Notice that the next day the professor's high-level goal may be the same near noon when he gets hungry, but he is likely to modify the plan based on his experience from the previous day. He may be inclined to return to the same restaurant if it was good, but may also want to sample others in order to learn whether others may be better (i.e., he may plan to try to learn more).

Clearly learning influences how we plan, and hence how the action hierarchy is formed and executed. For instance, as we learn the various routes to different restaurants, we essentially develop a "cognitive map" of the streets to get to the restaurants, and we use this map in the future (e.g., we plan over that map to minimize our travel time). We think of this "map" as a type of model that we learned that allows us to *predict* how a variety of plans will work, and hence it allows us to optimally achieve our goals by choosing the plan that minimizes travel time. This feature of exploiting past knowledge to predict (plan) ahead, and the process of choosing the "best plan," are essential features of successful planning, and flexible intelligent behavior. Moreover, the focus of attention is essential to planning, both the "internal focus" on traversing the action hierarchy, and the focus during execution of the plan to detect plan failures (e.g., observing a blocked street). Due to the hierarchical nature of the process, it seems that both planning and attention have hierarchical characteristics, and there is neurophysiological evidence of this intuition.

Optimization is essential to choose which plan is best.

6.1.2 Generic Planning Steps

While the above example serves to illustrate some of the essential features of how a human plans in one situation, it is useful to consider the following generic planning steps:

1. *Represent the problem ("planning domain"):* In order to plan, you must have some type of representation (model) of the problem that must be solved. This model could be in the form of a road map if you are trying to plan a route to get somewhere, or it could be a more conceptual map of the structural-functional characteristics of a problem. We generally think of these models as being acquired via experience (i.e., via learning), however, it is certainly the case that instincts (models passed to us via evolution) affect planning. For instance, we have certain "hard-wired" knowledge that can be thought of as aspects of models that influence planning (e.g., tendency to have a fear of snakes and some insects). Our performance in planning is critically dependent on our model of the problem. A poor model will generally lead to a bad plan, or at least to one that soon fails, thus requiring replanning. A high quality model that allows us to project far into the future (or down a hierarchy of tasks and subgoals), may lead to better plans. However, characteristics of the problem domain may make it impossible to specify a good model. For instance, time varying and stochastic features of some problem domains may make it impossible to predict into the future with any accuracy, and hence make it a waste of time to predict too far into the future. The difficulties in developing or generating a model include many of the same ones discussed in Part I for design and truth models. Differences arise however, since in planning we often learn the model as we plan.

2. *Set goal:* Setting goals is essential to planning, since without goals there is no purposeful behavior. Goals can be very different for different people,

environments, and times. Goals are driven by evolutionary characteristics (e.g., the goal of survival, the goal of reproduction), but in humans such goals can also be significantly affected by our values and ideals (e.g., ones set by culture). Goals can be learned, and can consist of a time-varying hierarchy or sequence of subgoals.

3. *Decide to plan:* Sometimes humans simply react to situations without considering the consequences of their actions. Other people decide to develop a plan since they may think that this will allow them to more successfully reach their goals. There are many issues that affect the decision of whether or not to plan (e.g., physiological and cultural). Many lower animals (e.g., some bacteria) cannot plan; they simply react to stimuli.

4. *Build a plan (select a strategy):* Normally the selection of a plan first involves projecting into the future using a model (e.g., in path planning on streets), and often involves considering a variety of sequences of tasks and subgoals to be executed (as in the action plan discussed above). In terms of a graph-theoretic view, you may think of this as a "tree" of plans where the nodes of the tree are tasks or subgoals, and links between these indicate plans (a path in the tree is a candidate plan). See Figure 6.2. How "deep" a tree to generate (e.g., how far to plan into the future) depends on the quality of the model, characteristics of the environment, and how much time or resources you have to plan. The second key component of selecting a plan is the solution of an optimization problem. For instance, suppose that the links on the "tree" that represents the set of possible plans are each labeled with integer values that represent the "cost" of performing the task represented by going in that direction in the tree. For instance, the cost may represent distance traveled or time to execute the task, and the characteristics of the cost are typically dictated by the goal. Next, suppose that the tree represents a finite number of possible plans, and that the cost of a plan is represented by summing the costs of each link that represents a step in the plan. We can then order the plans according to cost and perform minimization by picking the lowest cost plan (the "best" plan). Again, see Figure 6.2. For example, this may be the shortest route to the restaurant in the above example, if we are solving the subtask of route planning to the restaurant.

It is useful to view plan generation as forming a "tree" of possible behaviors for each plan. Plan selection involves ranking the quality of the behaviors and choosing the plan that produces the best behavior according to the model.

5. *Execute plan, monitor, and repair/replan:* After selecting a plan you must decide how to execute that plan. While we execute the plan, we monitor it by detecting deviations from what is expected to make sure that all is going well. Then, especially in an uncertain problem domain, it could be that there is a "plan failure" so that there is a need to repair the current plan, or to develop a completely new plan (the frequency of replanning is generally proportional to the amount of disturbances you have in the plant). The decision of whether to simply "tweak" the current plan, or develop a completely new one is difficult and can involve assessments of available resources (e.g., time), and the extent to which goals are being

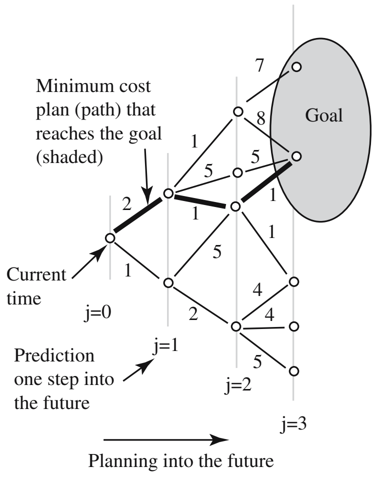
Minimum cost plan (path) that reaches the goal (shaded)
Goal
Current time
j=0
j=1
j=2
j=3
Prediction one step into the future
Planning into the future
7
8
1
5
5
2
1
1
1
5
1
4
4
2
5

Figure 6.2: Tree representation of the alternative plans that can be considered at some point in time, along with the costs of executing such plans.

met. Some problem domains are particularly difficult to monitor and hence there may need to be a parallel process operating that estimates the "state" of the domain from available sensed information (this is sometimes called "situation assessment"). Our ability to do this depends on the "observability" properties of the problem domain (i.e., whether we can compute the state of the plant from measured inputs and outputs). When using such estimates, you may need to *guess* whether the plan is succeeding and subsequently replan.

Next, it is important to note that there are many cognitive factors that can influence how we plan. For instance, the amount of knowledge we have and our ability to learn is critical. Our current stress level, emotions, coping skills, personality, values, and self-confidence all affect our performance in planning. Moreover, the capabilities of our biological neural network in working memory affect the complexity of plans we can consider, and rate at which we can develop plans. Our attentional skills play a key role in ensuring that we stay focused on our goals, and on the most important planning task at hand.

6.2 Design Example: Vehicle Guidance

In this section we will develop a simple planning strategy for control of the position of an autonomous vehicle to move it toward a goal position (i.e., to guide the vehicle). This example only illustrates the first of several ways in which we use planning concepts for control in this book. It is primarily used to give intuitive insights into how planning strategies operate. In the next section we will explain more advanced concepts on how to design planning strategies for nonlinear dynamical systems. In Section 9.4.5 we will discuss how learning and planning can be combined in adaptive control. In Chapter 16.5 we will use basic ideas from planning systems to formulate an approach to evolutionary adaptive control. Finally, in Section 19.6 we will discuss how biomimicry of learning and planning of social foraging animals can be used in distributed coordination and control for vehicles.

6.2.1 Obstacle Course and Vehicle Characteristics

The particular type of vehicle guidance problem we will consider will be one where we seek to guide the vehicle from some initial position to a goal position while avoiding collisions with obstacles. For example, you might think of trying to guide a vehicle through the halls of your building without running into walls. We will assume that we have *perfect* information about where obstacles are, and for convenience we assume that the vehicle is in a rectangular room and that the obstacles are poles with known (x, y) positions. In particular, we consider a room such that the x-coordinate, $x \in [0, 30]$, and the y-coordinate, $y \in [0, 30]$, with the poles shown from a top view in Figure 6.3. We assume that the initial vehicle position is $(5, 5)$ and that the goal position is $(25, 25)$ as shown in the figure via the square and "$\times$" respectively.

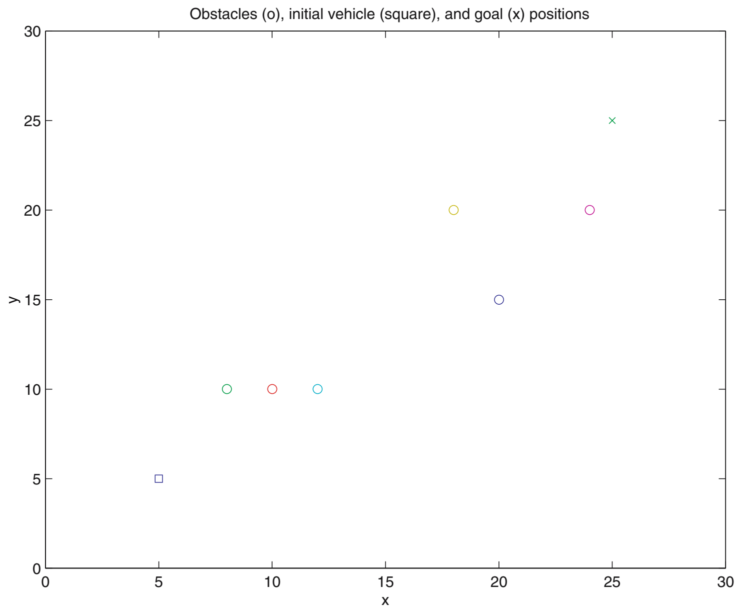

Figure 6.3: Initial vehicle position, goal position, and obstacles.

We assume that our vehicle is as shown in Figure 6.4 and that each side of the cubical vehicle measures 2.5 units so that it cannot fit in between the three poles shown in Figure 6.3 that are at positions $(8, 10)$, $(10, 10)$, and $(10, 12)$, but it can fit in between the other obstacles. We assume that the vehicle knows its own position (e.g., via an overhead computer vision system) and the goal position that it seeks to move to.

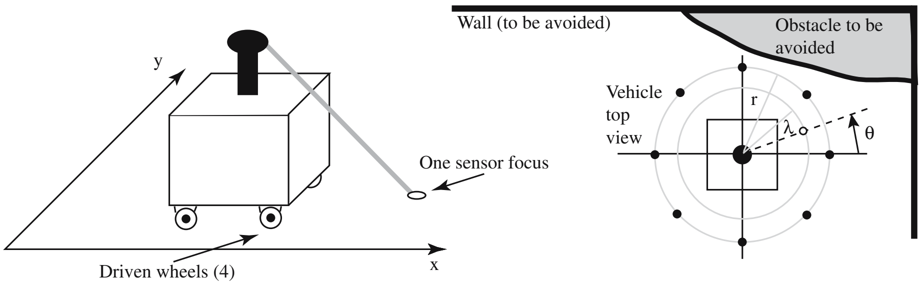

Figure 6.4: Autonomous vehicle guidance problem.

We will essentially ignore vehicle dynamics and assume that when a vehicle decides to move from one position to another position, it can approximately do so in one time step (but we do not put explicit units on distance or length of the time step). The "approximate" part is due to the fact that we assume it may not reach the precise desired position (e.g., due to inaccuracies in the vehicle drive system). In particular, if the vehicle's current position is $(x(k), y(k))$ and the onboard computer commands it to move at an angle θ a distance of λ (see Figure 6.4), it does so according to

$$\begin{bmatrix} x(k+1) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \end{bmatrix} + \lambda \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} + \Delta\lambda \begin{bmatrix} \cos(\Delta\theta) \\ \sin(\Delta\theta) \end{bmatrix}$$

where the sum of the first two terms on the right side of the equation represent the desired position. Here, we choose $\lambda = 0.1$. The last term is a noise term that represents effects of uncertainty that result in the vehicle not perfectly achieving the desired position. We choose $\Delta\lambda$ to be a random number chosen at each time step uniformly on $[-0.1\lambda, 0.1\lambda]$ representing that there is a 10% uncertainty in achieving the commanded radial movement. Also, we assume that $\Delta\theta$ is uniformly distributed on $[-\pi, \pi]$. Hence, when the vehicle is commanded to go to a particular position in one time step, all we know is that it ends up somewhere in a circular region of radius 0.1λ around the desired position. Notice that in order to make such movements, the vehicle needs to sample its own position at each time step. Hence, feedback control is used in the following way for guidance: the current position is sensed, and the command is made to move the vehicle to the new position. The vehicle may not end up where it was commanded to go, but at the next time instant, we will sense the vehicle's position and make adjustments from that point, and so on.

6.2.2 Path Planning Strategy

The assumption that we know exactly where all the obstacles are greatly simplifies the planning problem and allows us to focus only on some basic features of planning strategies; later we will remove this assumption and discuss how a vehicle could learn where obstacles are, plan based on that information, and even cope with moving obstacles. It should be clear that since we assume that we know where the vehicle and all the obstacles are, there is no need for a sensor that measures proximity to, or characteristics of obstacles. In a certain sense we have a perfect model of a *part* of our environment. We do not have a perfect model of the entire environment due to the uncertainty in reaching a desired commanded position that was discussed above.

Obstacle and Goal Functions

How can we represent and utilize the information given in Figure 6.3 about where the vehicle starts, where it should go, and where the obstacles are? First, since we are using a planning strategy, it is critical to realize that we need to formulate the path-finding problem as an optimization problem. To do this, we take the simple approach of constructing a surface (sometimes called a “potential field”) that represents where the obstacles are. In particular, to represent the obstacles in Figure 6.3, we take Gaussian functions of unity height and center them at each of the obstacles and compute an “obstacle function” $J_o(x, y)$ that is the *maximum value* of each of those functions at each point (x, y) as shown in Figure 6.5 (the use of the maximum of the six Gaussian functions representing the six obstacles, rather than, for instance, simple addition of the six Gaussian functions, ensures that each obstacle position is represented independent of the others). In Figure 6.6 we show the contour plot of $J_o(x, y)$ along with the initial vehicle position and goal position. The contour nicely shows the “spreads” (variances) of the Gaussian functions and that there is a type of overlap such that values of $J_o(x, y)$ are at least a bit above zero for any position where the vehicle should not be in order to avoid collision with obstacles. Also, we will scale the obstacle function with a positive constant $w_1 > 0$ in our planning strategy below; however, here we choose $w_1 = 1$. Note that if you moved the vehicle about the environment in a way that the vehicle position is moved to points that try to *minimize* $J_o(x, y)$ (e.g., via hill climbing), then the vehicle will avoid the obstacles, due to the tails of the Gaussian functions. For many vehicle initial positions, the vehicle would move to the edge of the region, and when it arrives there, we always keep it on the edge.

Next, we show how to represent the goal of being at the position $(25, 25)$. To do this, suppose that we think of penalizing not being at this position by placing the minimum point of a quadratic (bowl) function

$$w_2 J_g(x, y) = w_2 \left[[x, y]^\top - [25, 25]^\top \right]^\top \left[[x, y]^\top - [25, 25]^\top \right]$$

where $w_2 > 0$ is a scale factor we choose as $w_2 = 0.0001$ (we will explain this below) that will multiply this function. The scaled function is shown in

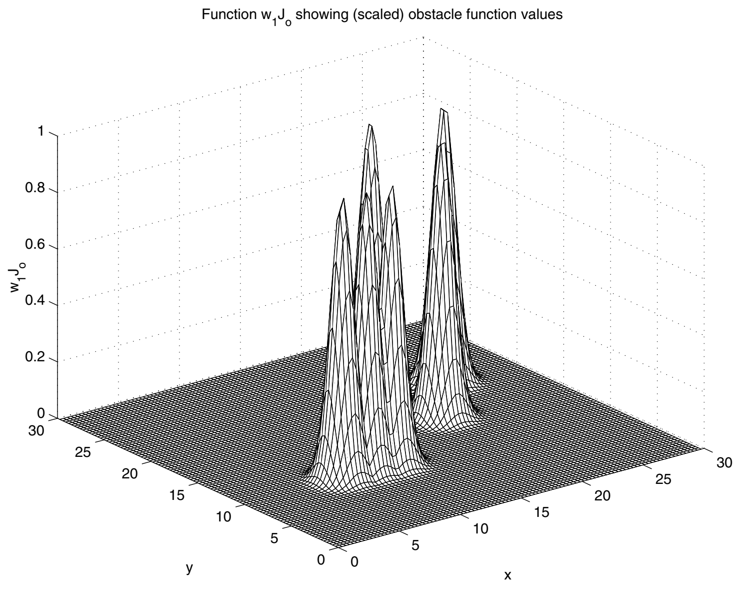

Figure 6.5: Obstacle function $J_o(x, y)$ (scaled by w_1).

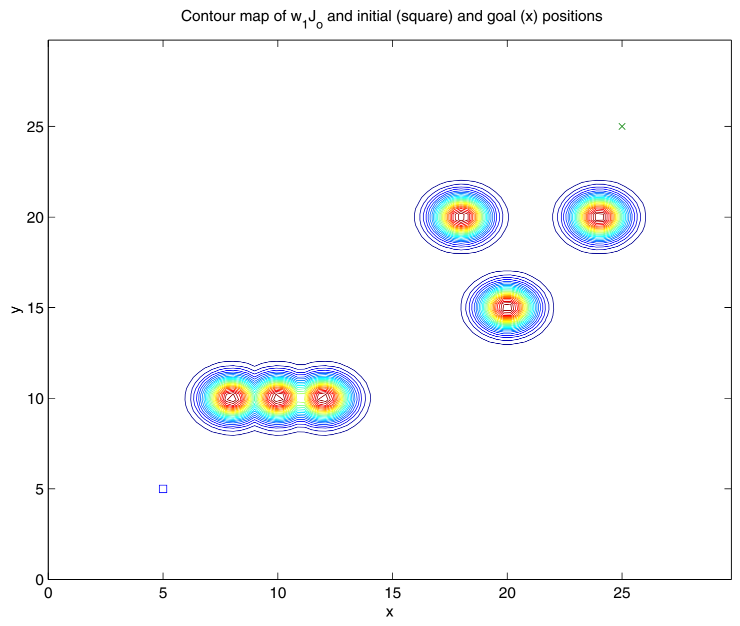

Figure 6.6: Obstacle function $J_o(x, y)$ (scaled), contour form, with initial vehicle position and goal position.

Figure 6.7 as a contour plot. If at each time step the vehicle moved to go down the surface, it will move toward the goal, but it may run into an obstacle.

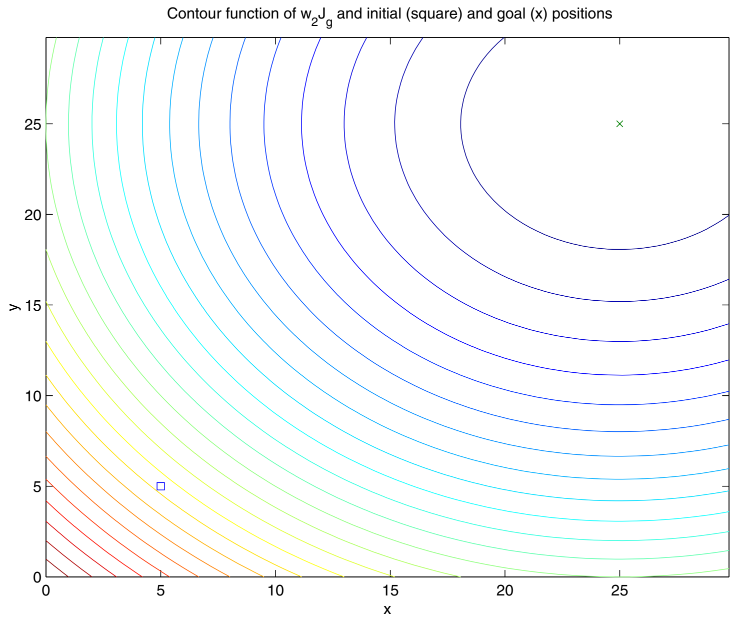

Figure 6.7: Goal function $w_2 J_g(x, y)$, contour form, with initial vehicle position and goal position.

Plan Generation and Selection

How does the planning strategy generate, evaluate, and select plans so that it can select which direction to move? To explain this, we first form our cost function for the planning strategy.

Multiobjective Cost Function: From the discussion in the previous subsection, it should be clear that if you commanded the vehicle to move a distance of λ in a direction θ that is chosen by simply moving in the "direction of steepest descent" on the function $J_o(x, y)$, then the vehicle would avoid obstacles but not reach the goal position and stay there. Similarly, if the direction was chosen to be the one with steepest descent for the $J_g(x, y)$ function, then it would move to the goal position but may collide with some obstacle for some initial vehicle positions.

Multiple goals can be represented by a multiobjective cost function.

To solve this problem we will use a "multiobjective cost function" (actually a special case where a "scalarization" approach is used to form a multiobjective cost, which is one of many ways to generate a Pareto cost)

$$J(x, y) = w_1 J_o(x, y) + w_2 J_g(x, y)$$

shown in Figure 6.8 where the weights w_1 and w_2 specify the relative importance of achieving obstacle avoidance and reaching the goal (but you must take into consideration the magnitudes of the values of each term in selecting these). Our choices of the weight values above represent that obstacle avoidance is important, but you must also keep moving toward the goal position. The choice of the weights will affect the shape of the trajectory that the vehicle will move on toward the goal position.

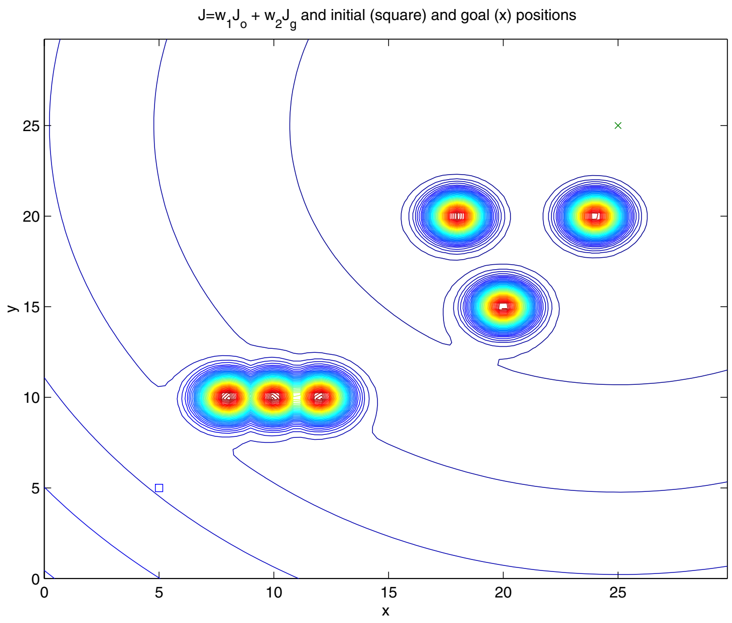

Figure 6.8: Multiobjective cost function $J(x, y)$ for evaluating plans.

Plan Generation and Selection: We take a very simple approach to plan generation and evaluation. If the vehicle is at a position (x, y), we simply compute the value of J at N_s values (x_i, y_i), $i = 1, 2, \ldots, N_s$, regularly spaced on a circle of radius r around the vehicle position (see Figure 6.4, where we have $N_s = 8$). Here, we will use $r = 1$ and $N_s = 16$. This generates 16 plans, where we "predict" one step ahead (clearly we could compute more values of J that are along other longer paths). We view the set of plans as "the vehicle is at (x, y), move it to (x_i, y_i)." We choose the plan to execute by finding a value i^* such that

$$J(x_{i^*}, y_{i^*}) \leq J(x_i, y_i),\ i = 1, 2, \ldots, N_s$$

(i.e., by finding the direction which will result in minimization of the multiobjective cost function). We then call this direction $\theta(k)$ and command the vehicle to take a step of length λ in the direction $\theta(k)$.

Notice that the above approach will approximate the "steepest descent approach" (hill-climbing) discussed above but we do not need analytical gradient information since we do not explicitly compute the gradient of the multiobjective cost function. Higher values of N_s cost more computations in plan generation and evaluation, but they also provide for more precise directional commands. Notice that using the above strategy, we expect that for any initial position on Figure 6.8, the vehicle will navigate so as to avoid the obstacles and move toward the goal by simply moving down the surface. Finally, notice that there is nothing special about the circular "pattern" of points that are evaluated on the J function. Other choices could work equally well. In fact, in Part V we will consider many other choices for the pattern of points that are used in deciding which direction to move to find the minimum point of a function (e.g., via pattern search methods), some of which are motivated by how animals search for food (a goal).

6.2.3 Simulation of the Guidance Strategy

Using the planning strategy, obstacle course, and vehicle, we get the trajectory shown in Figure 6.9. Clearly, the vehicle moves so as to avoid the obstacles (via the effect of J_o) but tries to stay on course to the goal (via the effect of J_g). The effects of the uncertainty in reaching commanded positions is seen by the small deviations on the trajectory that are "corrected" at each step since we assume that the vehicle gets a measurement of its own position at each time step. Other vehicle paths result from other choices of obstacle and goal functions and their scale factors (e.g., for this example, higher weight on the goal function tends to reduce deviations away from obstacles). Moreover, a different pattern of points where the multiobjective cost function is evaluated can result in a different path. For instance, using fewer points on the circular pattern results in trajectories that are not as smooth.

6.2.4 More Challenges: Complex Mazes, Mobile Obstacles, and Uncertainty

In this section we have studied a highly idealized planning problem. For instance, the assumption of *perfect* knowledge of the obstacle positions will not hold in any real obstacle avoidance problem. Removing the assumptions can quickly complicate the use of planning strategies, as we will see next.

Dead Ends and Circular Loops

Above, our type of obstacle course is quite simplistic. In some environments it is better to think of the obstacle course as a type of complex "maze" with many possible paths, many of which may not lead to the ultimate goal position (i.e., there may be "dead ends" or circular loops). See Figure 6.10(a). Suppose that we use the same basic approach as for our obstacle course in Figure 6.3 where we place functions that indicate that we should stay away from obstacles.

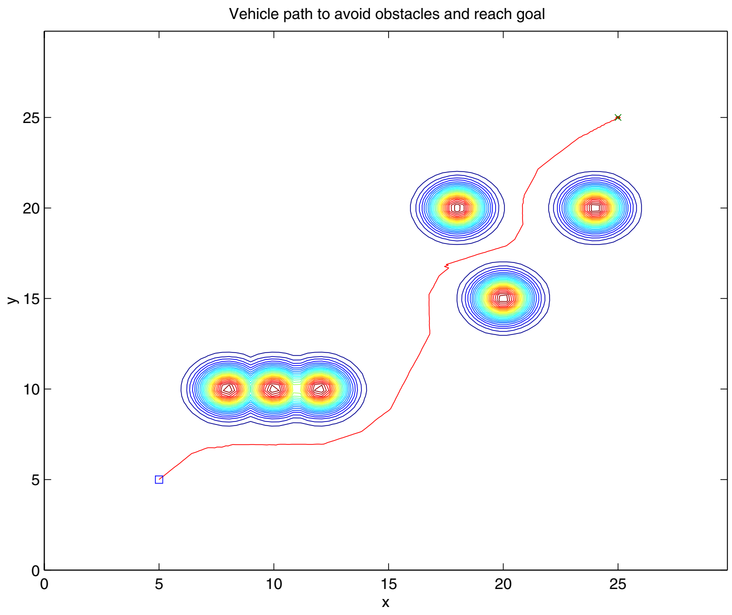

Figure 6.9: Vehicle path for obstacle avoidance and goal seeking.

How? While you could invent many types of functions, you could simply use a fine grid of appropriate Gaussian functions to get the proper shape for the $J_o(x, y)$ surface. A problem arises, however, with specifying the goal function and hence the multiobjective cost function. Suppose that we choose it as we did for the above example to be a simple quadratic function with a minimum point at the goal position. To see where the problem arises, suppose that we use the same planning strategy as in the previous subsection. In this case, it should be clear that for that obstacle course, with reasonable choices for the obstacle function and multiobjective cost function weights, the vehicle trajectory would move roughly diagonally (e.g., on paths 2, 3, or 4 in Figure 6.10(b)) toward the goal position similar to how it did in Figure 6.9 until it got to the curved wall in the "northeast" part of the maze. There, provided that r (the radius of the circular pattern of points where J is computed) is relatively small and we do not predict ahead more than one step, the vehicle will get stuck against the curved wall since it will listen to the goal function, but still try to avoid hitting the curved wall. It will get stuck at a "local minimum" on the multiobjective cost function. Notice that it does this even though if it could simply "see a little farther," it could navigate around the curved wall by going northwest, then back to the east to the goal position.

How can we solve this problem? One way is to use the a priori knowledge of the obstacle course and design the multiobjective cost function so that there is only one minimum, the global one, at the goal position. Another way is to design the obstacle and goal functions in a simplistic way as we did in the

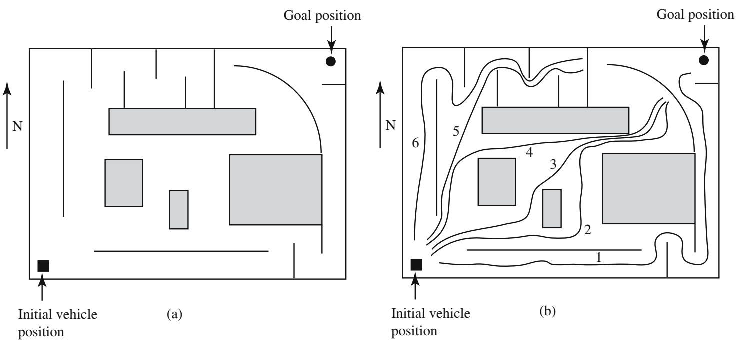

Figure 6.10: (a) Obstacle path viewed as a maze (notice dead ends), (b) Possible paths through the maze as computed by prediction in a planning strategy (numbered 1–6).

last section, then to exploit the "look-ahead" capabilities of planning strategies to find local minima on the multiobjective cost function that result in dead ends. To do this, suppose that at each step, the vehicle computer computes a tree of paths rooted at the current vehicle position (where it is assumed in the generation of that tree that the vehicle actually reaches the position desired on each move even though this will not be the case according to our model). For instance, suppose that it is constrained so that as shown in Figure 6.10(b), it computes six potential paths of the same length (length is not given by the physical length of the path, but by how many steps are taken, so in the figure each of the cases, 2–6, shows paths where the vehicle is stuck up against a wall for some time). It may come up with these potential paths by sampling the known multiobjective function, and some strategies even use minimization in the choice to limit the number of potential paths. For instance, in Figure 6.10(b) we show only six potential paths, not the many possible small deviations from these six. Next, we have to choose the best path. For this, we may use some method to detect when a plan will result in deadlock (no progress for a fixed number of steps), or we may try to minimize the number of required steps to get to the goal. The paths that are clearly unsuccessful can be eliminated from consideration and the first step suggested by the most successful plan can be taken. In the case where the maze is known perfectly and there is no uncertainty in reaching a desired position, there is no need for replanning at each step. You just follow the generated plan. However, in our model when we do not reach the commanded desired position, replanning (regeneration of plans and selection of new plans) is needed. How much replanning needs to be done? It depends on the magnitude of the uncertainty. Large uncertainty will lead to the need for frequent replanning.

Mobile Obstacles and Uncertainty

Next, note that if the obstacle environment is dynamic in the sense that, for instance, obstacles can move, our approaches require extensions. For instance, if some obstacle suddenly appeared at some position and we did not know about it, our vehicle can simply collide with it. Or if the walls and obstacles in Figure 6.10(a) moved in predictable ways, it should be clear that a "look-ahead strategy" may be needed. If the obstacles moved in unpredictable ways, then our model may not be able to accurately represent this so the vehicle will need to sense the environment while it navigates it and try to *learn* about obstacle positions and movements. Clearly this creates a very challenging obstacle avoidance problem.

6.3 Planning Strategy Design

Next, we distill the essential ideas from the psychology of planning in Section 6.1, some of which were explained via the path planning example of the last section, and show more clearly how they can be utilized in controllers for dynamical systems. Our focus here is on plants of the type that are typically considered in conventional control. First, we will think of planning systems as being computer programs that emulate the way experts plan to solve a control problem; notice the connection to how we thought of the heuristic design process for fuzzy and expert controllers. Note, however, there is an essential difference from how we thought of fuzzy and expert control: a planning system uses an explicit model of the plant. We will discuss several issues surrounding the choice of this model, plan generation, and selection. For simplicity, we will first ignore the hierarchical issues involved in planning and simply focus on how to plan at one "node" of an action hierarchy to achieve what might be a sequence of changing goals. Later in this section, however, we will discuss hierarchical planning.

6.3.1 Closed-Loop Planning Configuration

A generic planning system can be configured in the architecture of a standard control system as shown in Figure 6.11. In the context of human planning problems, the problem domain is the plant and environment. There are measured outputs $y(k)$ at step k (variables of the problem domain that can be sensed in real time), control actions $u(k)$ (the ways in which we can affect the problem domain), disturbances $d(k)$ (which represent random events that can affect the problem domain and hence the measured variable $y(k)$), and goal $r(k)$ (what we would like to achieve in the problem domain) which is called the reference input in conventional control terminology. There are closed-loop specifications that quantify performance specifications and stability requirements.

Planning (and replanning) often utilizes feedback to correct for prediction model errors.

The types of plants we consider in this section are those with

$$y(k+1) = f(x(k), u(k), d(k)) \tag{6.1}$$

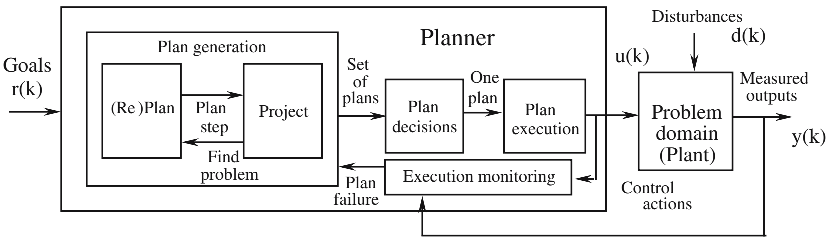

Figure 6.11: Closed-loop planning system.

where $y(k)$ is the measured output and f is a generally unknown smooth function of the state $u(k)$ and measurable state $x(k)$,

$$x(k) = [y(k), y(k-1), \ldots, y(k-n), u(k-1), u(k-2), \ldots, u(k-m)]^\top \quad (6.2)$$

Let

$$e(k+1) = r(k+1) - y(k+1)$$

be the tracking error. Generally, our objective here will be to make the tracking error as small as possible for all time, and in particular, we would like it to asymptotically approach zero so that the output follows the reference input.

Consider a plan to be a sequence of possible control inputs, where the i^{th} plan of length N at time k is denoted by

$$u^i[k, N] = u^i(k, 0), u^i(k, 1), \ldots, u^i(k, N-1)$$

Our objective is to develop a controller that is based on a planning strategy. To do this, we will use a model and an optimization method to evaluate the quality of each plan. This will provide a ranking of the quality of the plans. After that we will choose the plan that is best (call it plan i^*), and let the control input at each time instant k be

$$u(k) = u^{i^*}(k, 0)$$

That is, at each time k we choose the best plan $u^{i^*}[k, N]$, then use the first input from the control sequence as the input to the plant. The process is repeated at each time instant. Clearly, you could use a lower frequency of replanning, where, for example, you could generate a new plan every other sampling instant, and execute the first *two* inputs from the optimal plan each time.

6.3.2 Models and Projecting into the Future

Good models lead to good plans; bad models can lead to unstable behavior and poor performance.

There are a wide range of possibilities for the type of model that is used, and the type depends on the problem domain, the capabilities of the planner to store and use the model, and also the goals. For instance, a model used for planning could be continuous or discrete (e.g., a differential or difference equation), and it could be linear or nonlinear. It may be deterministic, or it may contain an explicit representation of the uncertainty in the problem domain so that plans can be chosen taking the uncertainty into account. In industrial practice, in the so-called "model-predictive control" (MPC) method, linear models are often used for the plant and this approach is considered in Design Problem 6.2.

Just like the design model used for control design, it will *always* be the case that the model will not be a perfect representation of the plant and the environment. This implies that there will always be uncertainty in planning, and hence there will always be a bound on the amount of time that it makes sense to project (simulate the model) into the future. Projecting into the future too far becomes useless at some point since the predictions will become inaccurate at some point, and hence provide no good information on how to select the best plan. The difficulty is knowing how good your model is and how far to project into the future. Finally, note that you may actually want your model to be able to predict what goal is going to occur in the future since in the formulation in this section we could have a time-varying goal. If the goal can be predicted, contingencies can be developed, and earlier plans may be modified to try to ensure success for not only the current goals, but anticipated ones.

No model is perfectly accurate; hence, predictions based on it are always in error.

Here, we use a general nonlinear discrete time model

$$y_m(j+1) = f_m(x_m(j), u(j))$$

with output $y_m(j)$, state $x_m(j)$, and input $u(j)$ for $j = 0, 1, 2, \ldots, N-1$. Notice that this model can be quite general if needed; however, in practice, sometimes a linear model is all that is available and this may be sufficient. Let $y_m^i(k, j)$ denote the j^{th} value generated at time k using the i^{th} plan $u^i[k, N]$; similarly for $x_m(k, j)$. In order to predict the effects of plan i (project into the future) at each time k you compute for $j = 0, 1, 2, \ldots, N-1$,

$$y_m^i(k, j+1) = f_m(x_m(k, j), u^i(k, j))$$

At time k to simulate ahead in time, for $j = 0$ you initialize with $x_m(k, 0) = x(k)$. Then, generate $y_m(k, j+1)$, $j = 0, 1, 2, \ldots, N-1$, using the model (note that you will need to appropriately shift values in x_m at each step) and generate values of $u^i(k, j)$, $j = 1, 2, \ldots, N-1$, for each i.

6.3.3 Optimization Criterion and Method for Plan Selection

Next, the set of plans (strategies) is "pruned" to one plan that is the best one to apply at the current time (where "best" can be determined based on, e.g., consumption of resources). Hence, optimization is central to the activity of planning (just as we will later see that optimization is central to the activities of attention, learning, evolution, and foraging). The specific type of optimization approach that is used for plan selection depends on the goals, cost function, and type of model that is used to predict into the future. For instance, if the model of the plant is a finite automaton the optimization problem can in some cases be formulated as a "shortest path" problem where you choose the plan (sequence of actions) that results in minimizing a cost function (e.g., the sum of the costs for the steps of a candidate plan) in a manner similar to how we choose the best plan for Figure 6.2. Such a shortest path problem can be solved with a number of methods. For instance, you could use dynamic programming or standard combinatorial optimization methods. Alternatively, when the state space is large it may be advantageous to use some "heuristic search" methods, such as the A^* algorithm. For more details on such approaches, see the "For Further Study" section at the end of this part.

Criteria for Selecting Plans

We need a criterion to decide which plan is the best. Here, we will use a cost function $J(u^i[k,N])$ that quantifies the quality of each candidate plan $u^i[k,N]$ using the f_m model. First, assume that the reference input $r(k)$ is either known for all time, or at least that at time k it is known up till time $k+N$. Generally, you want the cost function to quantify over the next N steps how well the tracking objective is met. One cost function that we could use would be

$$J(u^i[k,N]) = w_1 \sum_{j=1}^{N} \left(r(k+j) - y_m^i(k,j)\right)^2 + w_2 \sum_{j=0}^{N-1} \left(u^i(k,j)\right)^2 \tag{6.3}$$

where $w_1 > 0$ and $w_2 > 0$ are scaling factors that are used to weight the importance of achieving the tracking error closely (first term) or minimizing the use of control energy (second term) to achieve that tracking error. Other cost functions could use the output of a "reference model" as we do for several adaptive control approaches in Part III (see "For Further Study" for more details), an error measure on the other past values of the inputs and outputs, or an error measure on some other system variable. The choice of the cost function for evaluating the quality of the plans is application-dependent. To specify the control at time k you simply take the best plan, as measured by $J(u^i[k,N])$, and call it plan $u^{i^*}[k,N]$ and generate the control using $u(k) = u^{i^*}(k,0)$ (i.e., the first control input in the sequence of inputs that was best).

Note that this specifies a variety of methods to achieve what is called "model predictive control" (or "receding horizon control") in conventional control theory. Clearly, different models, cost functions, and optimization methods will lead to different closed-loop system performance characteristics. It can be difficult to know which optimization method to choose for a particular application. Often, however, practical aspects of the problem govern many aspects of the choice as we discuss next.

Nonlinear Optimization for Plan Selection

The challenge is to pick an optimization method that will converge to the optimal plan, and one that can cope with the complexity presented by the large number of candidate plans. Why is this a "challenge"? First, focusing on the complexity aspect, note that the inputs and states for the plant under consideration can in general take on a continuum number of values (i.e., an infinite number of values, with as many possibilities as there are real numbers), even though in particular applications they may only take on a finite number of values. This is the case in analog control systems even considering actuator saturation. For digital control systems you may have a data acquisition system that results in a certain quantization and hence, theoretically speaking there are a finite number of inputs, states, and outputs, for the model specified by f_m since it is typically simulated on a digital computer. However, this number can be *very* large! There is then, in general, an infinite (continuum) number of possible plans that you must compute the cost of (in a brute-force approach) in order to form the ranking of plans according to cost, and select the best plan.

But, in the conventional model predictive control approach that is widely successful in industrial applications, this problem has been solved. How? There, most often linear plant models are used, a manageable size is chosen for the prediction horizon N (and perhaps a longer sampling interval is used for the model, than for the digital controller), and it then becomes feasible to specify an *analytical* solution to finding the optimal plan (sequence of inputs). The optimization approach can actually choose the best plan from the infinite set of plans. That analytical solution is the so-called "least squares" solution that is only possible due to the use of the linear model. But, of course, no real plant is linear (even though it may act as though it is almost linear in some situations).

What if the nonlinear and uncertain characteristics dominate to the extent that a linear model is not sufficient for generating plans? Then, we could use a nonlinear model in the planner and try to employ some type of nonlinear optimization method where the "parameters" that are adjusted by the optimization method are ones that parameterize the infinite set of possible plans. Practically speaking, however, this can become problematic since if you use a nonlinear model for plan generation, you are confronted with a nonlinear optimization problem for which there is generally no analytical solution. There are, however, many algorithms that one could employ to try to solve this problem (e.g., steepest descent, Levenberg-Marquardt, etc., that are discussed in Part III). The problem is that none of these methods *guarantees* convergence to an optimal plan. They could even diverge and provide no solution, but typically they will converge to a local minimum. The plan that results from such a nonlinear optimization process cannot then be guaranteed to be the optimal one, and closed-loop performance can suffer. Having said all that, it is worthy to note, however, that in some practical industrial problems, engineers have managed to develop effective solutions via such a nonlinear optimization approach.

Nonlinear or combinatorial optimization can be used for plan selection.

Brute-Force Approach to Plan Selection

Next, suppose that you do not want to take the standard nonlinear optimization approach, yet you want to use a nonlinear model since its use seems essential to represent the salient features of your plant. Is there another approach? One standard approach is to discretize the input, state, and output spaces, generate all possible plans and compute the cost of each of them explicitly (sometimes it is even possible to simultaneously generate plans and evaluate costs, and thereby greatly reduce the number of potential plans since ones that are of very high cost may not need to be generated). Creating such a discrete model is not a trivial exercise since you want it to be not only discrete in time, but also in space. The discretization typically (virtually always) leads to the creating of a less accurate model so that in taking this approach you are trading off complexity management and optimization ease with accuracy in evaluating the plans. Also, unless you use a very coarse quantization you may still end up with too many plans to consider. Why? Suppose that there are N_u possible input values obtained via discretization, and that the model is deterministic so that one control input leads to only one possible state, then there are

Creation and evaluation of all possible plans is often computationally prohibitive.

$$(N_u)^N$$

possible plans at each time k. Suppose that we simulate ahead in time $N = 100$ steps, and $N_u = 1000$ (not unreasonable considering the types of levels of discretization that could be accurate for many plants). Clearly, due to the exponential growth in the number of plans, we can quickly encounter problems with computational complexity if we take the brute-force approach of generating all possible plans. Moreover, even if we generate all the plans, we will also have to evaluate the cost of each one. And, this must be done at each sampling instant. Having said all that, however, there are classes of problems where a discrete model provides a reasonably good representation of the plant, even with a small N_u, and sometimes only a small N is needed to evaluate the quality of a plan. In this case, the brute-force approach may work very well. Besides, specific application-dependent characteristics often allow you to "prune" the tree of possible plans. For instance, if you have rate constraints on your plant, then typically for every state only certain inputs are possible, since the input cannot change too much from what it was the last time. Moreover, sometimes coarser quantizations in time and space may work adequately for some plants.

There are ways to trade off computational complexity for the quality of plan selection and ultimately, performance.

6.3.4 Planning Using Preset Controllers and Model Learning

Next, we will discuss another approach to solve the complexity and optimization challenges involved in plan generation and selection. This approach can be thought of as a method to prune the tree of possible plans that is generated at each sampling instant.

Planning Using Multiple Controllers

Consider a specific controller (a "preset" controller) applied to the current state and reference input to be a type of "plan template" in that it specifies one way to respond for a sequence of times into the future, but the precise manner in which it generates inputs depends on what occurs over time as the plan is implemented. There is an analogy with how humans plan. In some problem domains we may have learned a finite set of possible *approaches* to solve a problem and we start solving it, picking what seems to be the best approach at each step.

Suppose that there are S such plan templates, which have the form of functions F_u^i

$$u^i(k,j) = F_u^i(x(k,j), r(k+1)) \; i = 1, 2, \ldots, S$$

where we assume we can measure $r(k+1)$. Hence, at each step we take each of these S plans and project into the future how each will perform, pick the best one, then let the control input be $u^{i^*}(k,0)$ where i^* is the best plan as measured by some cost function. For some practical applications the value of S need not be too large, and hence, if we take the "brute-force" approach of the last section, we overcome the problems discussed there in complexity and optimization.

In a related approach, it is also possible to use planning systems as general supervisory controllers in a similar manner to how expert controllers are used for supervision. In this case, the planner will, for instance, coordinate the use of a set of controllers where different controllers are used for different operating conditions. We will discuss such methods in Sections 9.4.5 and 16.5.

Planning Using Multiple Models or Tuned Models

Suppose that upon entering some problem domain you know that it is best modeled by one of S models that you have learned. Suppose that as you begin taking actions in the problem domain, you gather information that tells you which model is most appropriate at the current time. If you enter it at a different time, a different model may be more appropriate. Also, some environments are dynamic in that their characteristics change over time so that as you are taking actions in the domain with one model, you continually monitor the quality of the predictions it makes, and if appropriate, you can switch to another model. How do you plan with the model possibly switching at each time? You can do it just the same as discussed above. You simply change the model that you predict with over time. You can think of this as *learning* the appropriate type of model and using it to plan (the optimization method employed to select the model is implementing a type of learning).

Other planning systems may perform "world modeling," where a model of the problem domain is developed or modified (tuned) in an online fashion (similar to online system identification), and "planner design" uses information from the world modeler to tune the planner (so that it makes the right plans for the current problem domain). The reader will, perhaps, think of such a planning system as a general adaptive controller. It integrates learning of models directly into the planning process, in a manner reminiscent of how humans learn while planning. While we will not illustrate the operation of such strategies in this chapter, in Part III and Part IV we will discuss how to use such strategies in adaptive control.

6.3.5 Hierarchical Planning Systems

First, suppose that there is a hierarchy of models available for generating plans. To provide a simple illustration of some key ideas in hierarchical planning, suppose that we are performing route planning for a mobile robot at an industrial complex that has several buildings. Moreover, suppose that we organize our planner according to the description of the hierarchy in Figure 1.11 where we have a higher-level management level, and lower-level coordination and execution levels.

Suppose that we have several models, detailed ones of each room in every building, maps of each building that simply show how the rooms are connected via hallways, and maps of possible connection routes between the buildings. Suppose that we want to plan how to move the mobile robots around the industrial complex. Suppose that at the highest level the human operator specifies that the robot should go to building 3, room 416, to deliver a part that is needed in some manufacturing process. A planner at the management level could generate a set of routes between buildings and pick the best one considering other traffic and minimization of time of travel. A planner at the coordination level could be used to plan how to move to the desired room once the building is reached, and the planner at the lower level could specify how to navigate the room.

There are other types of hierarchical planners that will use multiple planners at the coordination and execution levels. For instance, sometimes the goals specified by the human can be broken down into multiple sequences of tasks at the management level, each one representing a different way to reach the human-specified goal. One approach could be selected and passed to the coordination level. At the coordination level we could view the sequence of tasks chosen at the management level as a sequence of goals, and each planner at the coordination level may then develop sequences of operations to try to achieve those (sub)goals. Clearly this sets up a recursion and we can view the chosen coordination level as plans, and the execution level can view those as goals and develop plans to meet them. Implementation is achieved by executing the low level sequences that try to meet the subsubgoals, and thereby the subgoals, and hence the goal specified by the human.

There are many design issues involved in constructing such a hierarchical planning system. For instance, the accuracy of the models at the various levels and the form of the cost functions used will significantly affect the performance of the system. Computational complexity is affected by the choice of the planning horizons at the various levels, and the lengths of these horizons is in turn affected by the quality of the models we use in planning (and uncertainty in the environment). Moreover, one approach to coping with computational complexity in some planning applications is to split the planning problem into a hierarchical functionality, since sometimes this can simplify plan generation and evaluation. Finally, we note that it is possible to incorporate adaptation and learning into the planning processes at the various levels.

6.3.6 Discussion: Concepts for Stable Planning

It is possible to perform stability analysis of control systems whose controller uses a planning strategy; in such cases you may study, for example, convergence of tracking error. For instance, there has been extensive work on the study of stability conditions (e.g., in terms of horizon length) for conventional linear model predictive control methods. Moreover, there has been other work focusing on stability of planning systems for plants with a discrete event character (see Design Problem 6.3). These studies show that there are several essential characteristics that affect stability properties, several of which can be thought of in terms similar to the discussion in Section 6.2.4, where we discussed dead ends, circular loops, obstacle mobility, and obstacle position uncertainty:

Stability analysis of closed-loop planning strategies depends critically on model accuracy, plant uncertainty, and plant nonlinearities.

- *Model accuracy:* The accuracy of the model used to project into the future significantly affects the analysis. In most analysis it is assumed that a *perfect* model is known or that the model perfectly represents all possible ways that the plant will respond to inputs.
- *Navigating through uncertainty:* Your ability to achieve a goal state in a tree of possible paths that are simulated (e.g., as shown in Figure 6.2) depends on the uncertainty present in the plant. You can think of the uncertainty as a type of adversary, and that your objective is to keep moving in directions so that the uncertainty will not over time conspire to make it impossible for you to navigate to your goal state (in terms of Figure 6.2, the actual structure of the tree is random so at some points in time some paths may lead to the goal with a certain cost, while at other times the cost may increase/decrease, or may not even lead to the goal state). The planning strategy tries to navigate the tree in a way so that even though the plant may make unpredictable moves, it will not be able to make moves that will make it impossible to reach the goal. Clearly, the number of steps you project in the future can critically affect your ability to navigate through the uncertainty. If you do not look far enough into the future, for some plants it may be possible that you will enter a region of the state space such that the effects of the uncertainty dominate and there is no way to navigate out of that region and to the goal state (e.g., in Figure 6.2, note that there are some "dead-ends" in the tree that is shown). On the other hand, it may not make sense to project more than one or two steps into the future for some plants since longer projections may neither result in better plans, nor help navigate through the space.
- *Avoiding traps:* For some plants, without projecting far enough into the future, it may be possible to get "trapped" in a cycle where you repeatedly visit a finite sequence of states on a loop. Moreover, it is of course possible that such circular traps arise in a nondeterministic manner, essentially combining the concerns of the last point with those of this one (i.e., random dead-ends and cyclical traps can arise).

The above discussion is simply intended to provide the interested reader with some intuitions about some issues that significantly affect our ability to perform stability analysis of planning systems for some classes of plants. For further study on this topic, see Design Problem 6.3 and the "For Further Study" section at the end of this part.

6.4 Design Example: Planning for a Process Control Problem

In this section we will develop a planning strategy for a very simple yet representative process control problem. We begin by introducing the control problem and then we design and test a planning strategy.

6.4.1 Level Control in a Surge Tank

Consider the "surge tank," shown in Figure 6.12, that can be modeled by

$$\frac{dh(t)}{dt} = \frac{-\bar{d}\sqrt{2gh(t)}}{A(h(t))} + \frac{\bar{c}}{A(h(t))}u(t)$$

where $u(t)$ is the input flow (control input), which can be positive or negative (it can both pull liquid out of the tank and put it in); $h(t)$ is the liquid level (the output of the plant); $A(h(t)) = |\bar{a}h(t) + \bar{b}|$ is the cross-sectional area of the tank and $\bar{a} > 0$ and $\bar{b} > 0$ (their nominal values are $\bar{a} = 0.01$ and $\bar{b} = 0.2$); $g = 9.8$; $\bar{c} \in [0.9, 1]$ is a "clogging factor" for a filter in the pump actuator where if $\bar{c} = 0.9$, there is some clogging of the filter and if $\bar{c} = 1$, the filter is clean so there is no clogging (we will take $\bar{c} = 1$ as its nominal value); and $\bar{d} > 0$ is a parameter related to the diameter of the output pipe (and its nominal value is $\bar{d} = 1$). We think of all these plant parameters as being fixed (but unknown) for a particular surge tank; however, we could consider other values for these parameters and test the controller for these. This models the situation where you want to develop one controller for many different surge tanks.

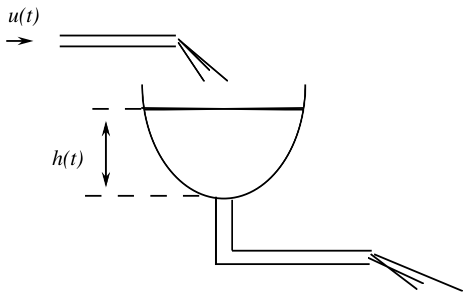

Figure 6.12: Surge tank.

Let $r(t)$ be the desired level of the liquid in the tank (the reference input) and $e(t) = r(t) - h(t)$ be the tracking error. Assume that you know the reference trajectory a priori and assume that $r(t) \in [0.1, 8]$ and that we will not have $h(t) > 10$. Assume that $h(0) = 1$.

To convert to a discrete-time approach, use an Euler approximation to the continuous dynamics to obtain

$$h(k+1) = h(k) + T\left(\frac{-\bar{d}\sqrt{19.6h(k)}}{|\bar{a}h(k)+\bar{b}|} + \frac{\bar{c}}{|\bar{a}h(k)+\bar{b}|}u(k)\right)$$

where $T = 0.1$. We assume that the plant input saturates at ± 50 so that if the controller generates an input $\bar{u}(k)$, then

$$u(k) = \begin{cases} 50 & \text{if} \quad \bar{u}(k) > 50 \\ \bar{u}(k) & \text{if} \quad -50 \le \bar{u}(k) \le 50 \\ -50 & \text{if} \quad \bar{u}(k) < -50 \end{cases}$$

Also, to ensure that the liquid level never goes negative (which is physically impossible), we simulate our plant using

$$h(k+1) = \max\left\{0.001, h(k) + T\left(\frac{-\bar{d}\sqrt{19.6h(k)}}{|\bar{a}h(k)+\bar{b}|} + \frac{\bar{c}}{|\bar{a}h(k)+\bar{b}|}u(k)\right)\right\}$$

Note that all the simulations in this section will include these constraints.

6.4.2 Planner Design

Here, for the sake of illustration we will use a nonlinear discrete-time model for the nonlinear discrete-time plant (the "truth model"). We will generate candidate plans using this model using the "preset controllers" approach discussed in the last section.

Taking the model of the last subsection as the truth model for the plant, the model that we will use in our planning strategy will have

$$A(h(t)) = \bar{a}_m (h(t))^2 + \bar{b}_m$$

with $\bar{a}_m = 0.002$ and $\bar{b}_m = 0.2$. For the model we use the same nonlinear equations as given in the last section, but we do not assume that we know the values of $\bar{c}$ and $\bar{d}$, so for these we use $\bar{c}_m = 0.9$ and $\bar{d}_m = 0.8$. It is interesting to note that if you plot the cross-sectional area of the actual plant, and the one used in the model, you get Figure 6.13, so you can see that they are somewhat different so that our model is clearly not the same as the plant (model).

So, is the model accurate enough to be used in projection? To answer this question we develop a simple controller and test it on the plant and controller. We use a proportional integral (PI) controller as the "plan template." In particular, if $e(k) = r(k) - h(k)$, we use

$$u(k) = K_p e(k) + K_i \sum_{j=0}^{k} e(j) \qquad (6.4)$$

Suppose that the goal is to get a reasonably fast response, with no overshoot in the tracking error $e(k)$.

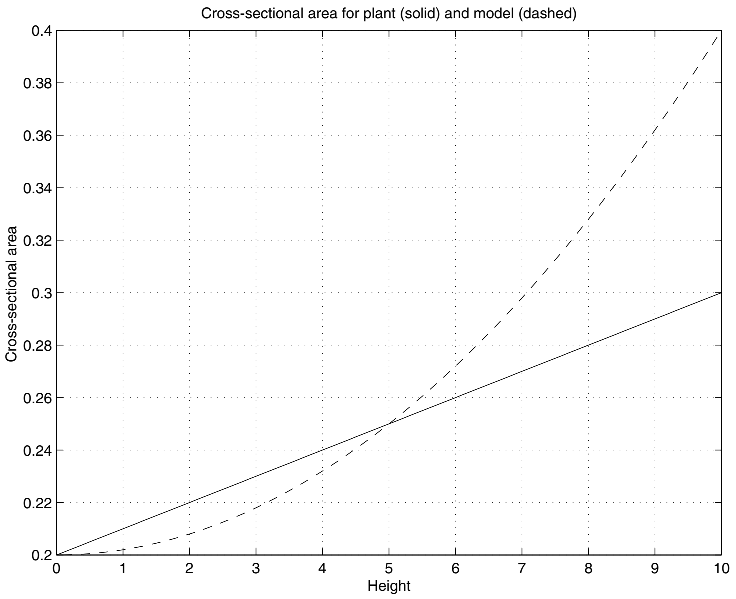

Figure 6.13: Cross-sectional area $A(h)$ for the plant (solid) and model to be used for projection (dashed).

Suppose that via experience in designing PI controllers for surge tanks with various cross-sectional areas, you know that typically

$$K_p \in [0, 0.2]$$

and

$$K_i \in [0.15, 0.4]$$

For instance, if you pick $K_p = 0.01$ and $K_i = 0.3$ and use the PI controller in Equation (6.4), you get the response in Figure 6.14. Notice that while the response is relatively fast, there is overshoot and that is undesirable.

You actually get a similar response if you use the same gains for the above model that will be used for projection in our planning strategy. To see this consider Figure 6.15, where we see that the difference between the regulated heights for the cases where we use the truth model for the plant, and where we use the projection model, are relatively small (there is more overshoot when the controller is used for the model rather than the plant). This gives us some confidence that our model is reasonably accurate; but of course to properly evaluate its accuracy, we need to consider how good a performance we can obtain when we use the model in a planning strategy for projection, and at the same time, use the truth model in the closed-loop.

We use the cost function in Equation (6.3) with $N = 20$ (for two seconds projection into the future), $w_1 = 1$, and $w_2 = 1$. Also, we assume at each

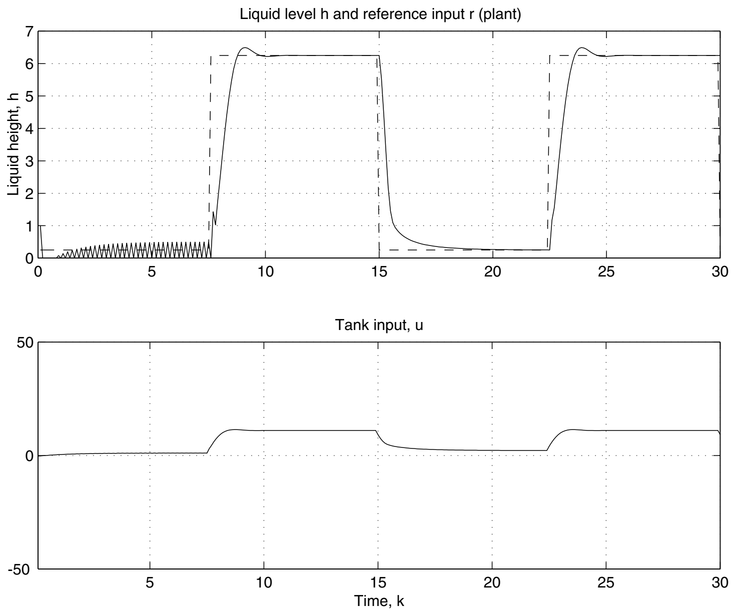

Figure 6.14: Closed-loop behavior of the surge tank using a PI controller.

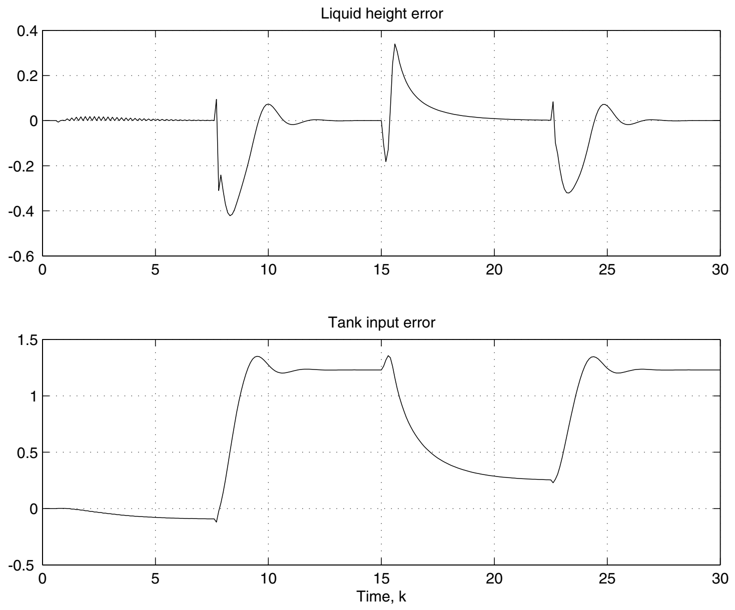

Figure 6.15: Error between cases where the truth model and projection model are used as the plant.

time instant that the reference input remains constant while we project into the future; this is equivalent to assuming that our evaluation of which controller is best is based on the reference input being constant.

Our plan templates are the PI controllers, with different values of K_p and K_i. In fact, we simply create a grid on the above ranges for the gains by considering all possible combinations of

$$K_p \in \{0, 0.05, 0.1, \ldots, 0.2\}$$

and

$$K_i \in \{0.15, 0.2, \ldots, 0.4\}$$

Hence, in this case there are $5 \times 6 = 30$ different plans (controllers) that are evaluated at each time step. To do this evaluation, we simulate using the projection model into the future two seconds for each PI controller. We initialize the simulations into the future with current error, and integral of the error.

6.4.3 Closed-Loop Performance

To see how the planning strategy operates, see Figure 6.16. Here, we see that we get a slower rise-time than in Figure 6.14 when we used the PI controller, but that we were able to tune the planning strategy (by adjusting w_1, w_2, and the grid on the PI gains) so that there is no overshoot, and still a reasonably good rise-time, and that was our main objective.

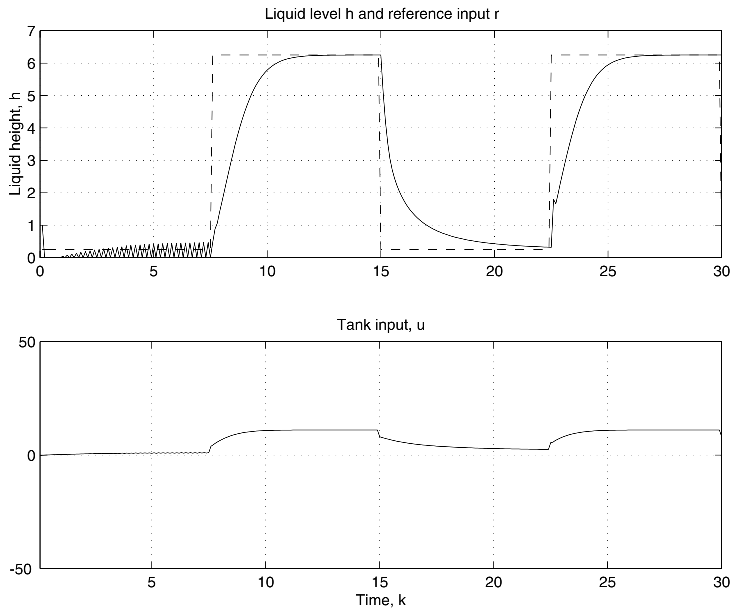

Figure 6.16: Closed-loop behavior of the surge tank using a planning strategy.

How does it achieve this performance? It switches controllers online and to see this, consider Figure 6.17. Note that we define the indices so that they are proportional in size to the K_p and K_i values (e.g., the $(1,1)$ controller has $K_p = 0$ and $K_i = 0.15$) so that it seeks to increase the K_p value to reduce tracking error and get a good rise-time, and lowers the K_i value to try to reduce overshoot. If you choose different values of the planning horizon N, you will get different switching sequences.

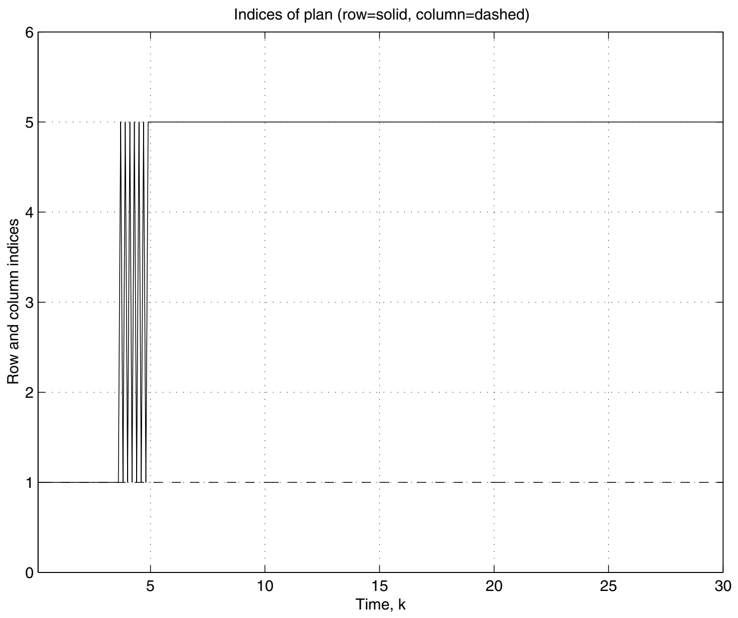

Figure 6.17: Indices of PI controllers that are used at each time step for the tank.

Similar performance to that shown above is found if you perturb some of the plant parameters. For example, if for the plant you let $\bar{c} = 0.8$ (representing more clogging), you get similar results to the above. Or, if you use the nominal value for $\bar{c}$ and use $\bar{a} = 0.05$, you get the cross-sectional area shown in Figure 6.18 and we get the closed-loop response in Figure 6.19.

Notice that while we still get an adequate rise-time, for this plant the planning strategy results in a small amount of overshoot; hence, you may want to tune the planner in order to improve the response. This shows that while the planning strategy may provide good performance for some plants, for some others the performance can degrade (not surprising). How robust is the controller to plant perturbations? It can be a challenging problem to design a single planning strategy that will perform adequately for all plants of a certain class (e.g., for a known set of structured perturbations about the nominal plant).

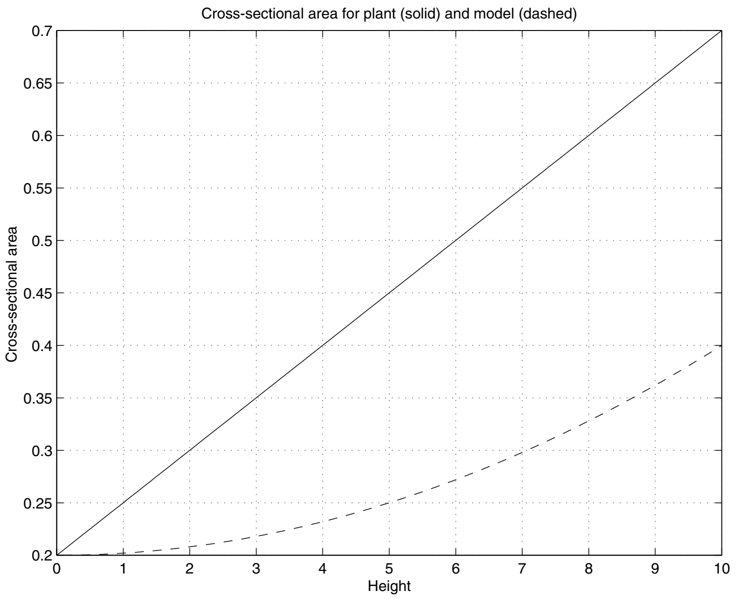

Figure 6.18: Cross-sectional area $A(h)$ for the plant (solid) and model to be used for projection (dashed).

6.4.4 Effects of Planning Horizon Length

Next, we return to using the parameters for the nominal plant and study the effect of changing the projection length N with all the same choices as in the previous subsection. In particular, we plot the tracking energy

$$\frac{1}{2}\sum_k (e(k))^2$$

and control energy

$$\frac{1}{2}\sum_k (u(k))^2$$

vs.

$$N \in \{1, 5, 10, 15, 17, 20, 25, 30, 33, 35, 36, 37, 38, 39, 40, 45, 50\}$$

as shown in Figures 6.20 and 6.21. This range of N was chosen by adding more points where the values of the tracking and control energy changed fast.

These plots show some justification for the choice of $N = 20$ in our earlier simulations. This choice did not cost too much computational complexity in projecting into the future, and yet gave a low tracking error (our main objective), with a reasonable amount of control energy. If you are not concerned about computational complexity, you may want to further increase the planning horizon, to get a similar value for the tracking energy, but with even lower

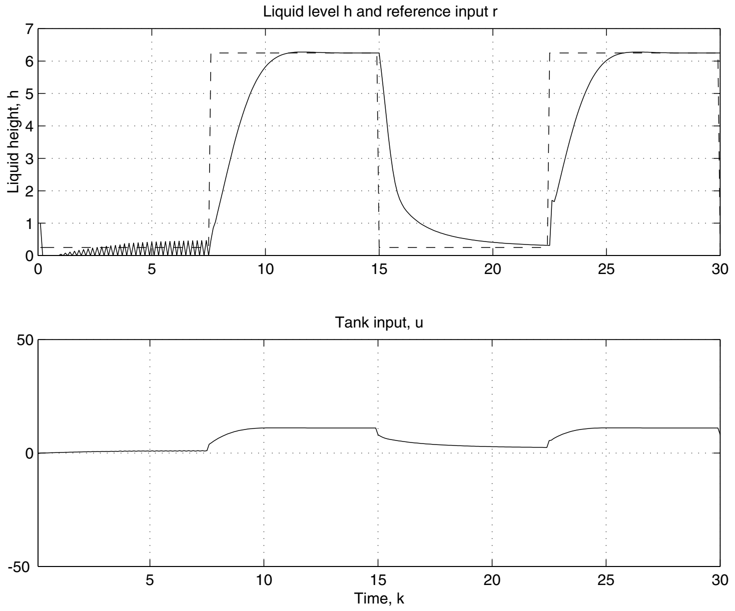

Figure 6.19: Closed-loop behavior of the surge tank using a planning strategy (different cross-sectional area).

control energy. Why did the values of the control energy change so quickly around the value of $N = 37$? Why does the tracking energy increase in the region from $N = 20$ to $N = 33$? Why is it the case that the control energy increases from $N = 1$ to $N = 10$? In general, how do you change the shape of the plots? Clearly, changing the w_1 and w_2 weights will change the shape, and hence, what choices you might make for what you call a "best" value of N. The model used for prediction, and the types of controllers that are simulated into the future will also change the shape. Moreover, the reference input can change it. Even though the generation of such plots can help you choose the planning horizon, it does not completely solve the problem. It simply provides insights.

Prediction horizon choice is difficult. Prediction too far into the future is computationally expensive and sometimes not useful due to plant uncertainty.

Finally, in some cases it is possible that longer planning horizons can actually *degrade* performance since the longer you simulate into the future with an inaccurate model, the less reliable the predictions tend to be. Hence, the optimization for plan choice can become inappropriate for selecting a good plan.

6.5 Exercises and Design Problems

Exercise 6.1 (Planning for Obstacle Avoidance):

(a) For the path planning problem in the chapter, use the simulation to generate plots that explain the effect of increasing the amount of uncertainty in where the vehicle ends up after a single step (i.e.,

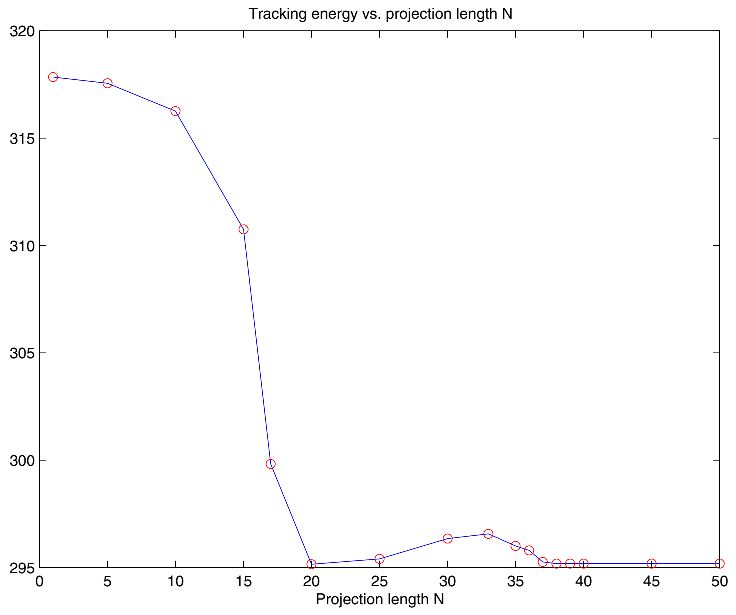

Figure 6.20: Tracking energy vs. projection length N.

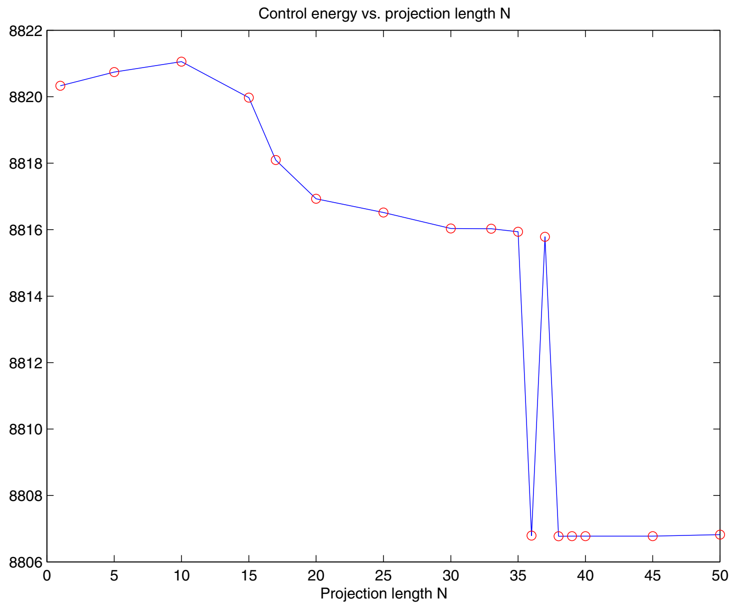

Figure 6.21: Control energy vs. projection length N.

explain the qualitative effects of the noise on the quality of planning). Can you choose the noise large enough so that the planning strategy fails to guide the vehicle to the goal after 500 steps?

(b) Next, study the effects of changing the value of r, the sensing radius. What happens if it is chosen smaller? Larger? If its value is too large, can the guidance algorithm fail? Illustrate your answer with a simulation.

(c) Illustrate the effects of changing N_s. What happens if $N_s = 4$? Illustrate with a simulation and explain how the movements of the vehicle change. What is the effect of using large values for N_s? Discuss smoothness of trajectories and computational complexity issues.

Exercise 6.2 (Model Predictive Control for a Simple Process Control Problem):

(a) For the MPC for the surge tank problem in the chapter, investigate the robustness of the strategy to measurement noise. To do this, you should precisely define what you mean by good performance, and investigate in simulation the effects of characteristics of noise (e.g., mean and standard deviation) on performance for a fixed MPC strategy.

(b) For the MPC for the surge tank problem in the chapter, investigate the robustness of the strategy to unknown characteristics of the tank cross-sectional area $A(h(t))$ (but for reasonable physical choices of the cross-sectional area). To do this, you should precisely define what you mean by good performance, and investigate in simulation the effects of characteristics of shape of the tank (e.g., if you characterize the shape with some nonlinear function, vary the parameters of the function) on performance for a fixed MPC strategy.

Design Problem 6.1 (Planning Ahead for Obstacle Avoidance):

(a) Simulate the obstacle avoidance problem in the chapter using the guidance algorithm defined there, but study a different placement of obstacles in the environment. Show a placement that the guidance strategy can successfully navigate, and one that it cannot successfully navigate.

(b) Repeat (a), but for a guidance strategy that predicts into the future multiple steps. Invent a placement of obstacles in the environment for which multistep prediction into the future allows successful navigation, where the strategy used in (a) does not. Hint: Consider a strategy that generates a tree of points with a root at the current position. For example, one approach would be to generate a circular pattern, pick the best point on that circle, then generate a circle of points around that point, and so on. The "best" plan is the path of best points found. You could experiment with different planning horizons and the frequency of replanning.

(c) Invent a placement of obstacles such that the strategy that you designed in (b) will fail in the sense that the robot will get stuck at a location other than the goal position. Redesign the look-ahead strategy so that it can successfully navigate it. Hint: Make the planning horizon vary with time in a way so that if it detects that it is "stuck," it lengthens its planning horizon until it finds its way around the obstacle (out of the local minimum). Illustrate the performance of the algorithm in simulation. Clearly explain your strategy and its operation. Discuss algorithm complexity.

Design Problem 6.2 (Model Predictive Controller Design for Tanker Ship Steering)*: In this problem you will study "model predictive control" (MPC) [192] for tanker ship steering. The tanker ship model that you will use as the truth model (to represent the plant) in all simulations should be the one given in Equation (4.5) that is simulated with a Runge-Kutta method in Section 4.3.1.

(a) For planning (prediction), use the linear model in Equation (4.4). Suppose this model is used with parameters specified for nominal conditions for the tanker ship; however, suppose that you use a discretized version of this model with a sampling period of $T = 1$. Hence, your discrete time model transfer function is

$$\frac{(-5.58e-05)z^3 - (5.635e-05)z^2 + (5.469e-05)z + 5.524e-05}{z^3 - 2.97z^2 + 2.939z - 0.9696}$$

which is obtained via a Tustin (bilinear) transformation when the nominal parameters ("ballast" conditions) are used and $T = 1$. Verify this. How accurate is this model? Simulate this linear discrete model and the nonlinear one. Highlight the similarities and differences in how the two models behave. In making this comparison induce disturbances in the nonlinear model (e.g., weight changes, wind, sensor noise, and speed variations) and explain how the plant differs from the linear discrete model in each case.

(b) Next, develop a method to project into the future and determine which sequence of inputs is best, then pick the first one to input to the plant for the current sampling instant. Let N denote the number of sampling instants that you simulate into the future. Suppose that you use a linear batch least squares approach (see Section 10.1 and in particular Equation (10.2)) to pick the best sequence of inputs to the plant, based on the linear discrete model. The key to solving this problem is to assume that the reference input trajectory is a constant, and to formulate the optimization problem as a linear least squares optimization problem. Show how to formulate the problem, and give an example where you code the example and actually find an optimal sequence of inputs to the plant. That is, simulate the closed-loop system when the MPC is used.

(c) Evaluate the performance of the MPC. Use similar reference inputs, and a similar sequence of investigations into the effects of disturbances, to what we did for the neural and fuzzy control methods. Develop a single numeric measure of performance (e.g., some quantification of tracking and control energy) and study how this measure changes for a range of values of the planning horizon N (make a plot). Repeat this for each disturbance condition. What is the best value to choose for N?

Design Problem 6.3 (Stability Analysis of Planning Systems)*: In this problem you will study stability analysis of planning strategies for two different plants that were studied in [196].

(a) Tank: Stability in the presence of uncertainty. Simulate, prove the strongest type of stability property possible.

(b) Load balancing in flexible manufacturing systems: Stability in the presence of traps. Repeat (a).