Algorithms for Planning under Uncertainty in Prediction and Sensing

Jason M. O'Kane Benjamín Tovar Peng Cheng

Steven M. LaValle

Department of Computer Science

University of Illinois at Urbana-Champaign

{jokane, btovar, pcheng1, lavalle}@cs.uiuc.edu

1 Introduction and Preliminaries

For mobile robots, uncertainty is everywhere. Wheels slip. Sensors are affected by noise. Obstacles move unpredictably. Truly autonomous robots (and decision-makers or agents in general) must act in ways that are robust to these sorts of failures and unexpected events which we may think of in general as *uncertainty*. In this chapter, we attempt to meet uncertainty head-on by explicitly modeling it and reasoning about it. We use the term *decision theoretic planning* to refer to this broad class of planning methods characterized by explicit accounting for uncertainty. We will consider a number of formulations for the problem of planning under uncertainty and present algorithms for planning under these formulations.

Uncertainty can take many forms, but for brevity and clarity we will restrict our attention to only two important types:

- **Prediction uncertainty** occurs when the effects of actions are not fully predictable. This can be thought of as an uncertainty in *future* states.
- **Sensing uncertainty** is uncertainty in the *current* state. This occurs, for example, in robots that have limited or imperfect sensing. We also admit the case where robots have no sensing at all.

Some systems can be adequately modeled without either form of uncertainty. Problems in this category can still be quite challenging and are the subject of many earlier chapters in this volume. Problems with only prediction uncertainty are addressed in Section 2. This manner of formulation is appropriate for robots in environments in which the effects of an action are not fully predictable, but with sufficient sensing capability to fully determine the effects of each action *a posteriori*. When a robot's sensors are no longer adequate to fully determine the current state, the problem moves from the familiar *state space* to a richer space called an *information space*. Formulations with sensing uncertainty – with or without prediction uncertainty – are the topic of Section 3.

In the remainder of this section, we discuss some preliminary ideas that are relevant no matter what sort of uncertainty is present.

Uncertainty as a game against nature A unifying theme will be the idea of uncertainty as a "game against nature." Imagine an external decision maker called *nature* whose decisions determine the values of all uncertain parameters. Executing a plan becomes an interaction with nature as well as with the environment. Both our robot and nature make decisions and the outcome is fully determined given both of these decisions. In a sense, we are pushing all of the uncertainty in a system off to nature. Then, if we can develop some model for how nature will make its decisions, we can build plans to react accordingly. We use the term *uncertainty model* for this description of how nature will make its decisions.

The uncertainty model we select will directly influence the solution concepts we use. That is, an uncertainty model determines the answer to the question "What does 'optimal' mean?". As a result, the mechanics of each planning algorithm will also change. In this chapter, we will consider two distinct types of uncertainty models:

- Under *nondeterministic* models [1, 2], uncertainty is expressed as a set of possible outcomes. This model is also sometimes called the "possibilistic", "worst case", or "set membership" model. Domains in which firm guarantees are required or that involve interaction with a strong antagonist are good candidates for nondeterministic uncertainty models.

- Under *probabilistic uncertainty* [3] we express uncertain events in terms of a conditional probability distribution over possible outcomes, given certain current conditions. This model is particularly well-suited for cases where uncertainty arises from precision errors in sensing or actuation, or from random

exogenous events.

The reader should note that legitimate criticisms can be leveled against both of these uncertainty models, some of which are elaborated in Section 2.1.2. Consequently, selecting an uncertainty model can sometimes be more of an art than a science. Most of the algorithms we will present are essentially independent of uncertainty model in the sense that they can be adapted to the type of uncertainty we select. Generally, we will derive similar but distinct versions for these two uncertainty models.

What is a plan? The concept of a solution for a planning problem in the absence of uncertainty is well understood: We seek a sequence of actions that transforms the system of interest from an initial state into a goal region, possibly optimizing some cost functional along the way. Uncertainty will force us to reconsider this notion of what a solution is.

Certainly the idea of a solution as a sequence of actions is made inadequate by the introduction of prediction uncertainty. Since state transitions are not fully predictable, we must prepare our agent to act in any state it may reach, rather than only those along a single path we have intended for it. Sensing uncertainty complicates the matter further because the agent will no longer even know its current state with certainty and instead must be able to react to any sensor/action history it encounters. These ideas will be made more formal in subsequent sections. The important idea here is that by allowing uncertainty we are forced to revise our notion of what constitutes a plan; for each new formulation we study, we will ask "What is a plan?".

Discrete vs. continuous spaces Many decision-theoretic planning algorithms are easiest to understand and implement under the assumption the spaces of states, actions and observations are finite, or at least countable. Indeed, we will adopt this assumption in our initial presentations of most techniques. However, in robotics, the most natural models often involve continuous spaces. For this reason, we must pay careful attention to how these methods can be used to deal with continuous-space problems. Any algorithm designed for a digital computer must have discrete versions of these spaces in some way. Such discrete spaces will generally fall into one of two broad categories:

- **Critical Events:** For some problems, there is a natural, finite partition of the state or action space into equivalence classes in such a way that the planning problem can be solved by considering only

these equivalence classes, rather than individual states or actions.

- **Sampling:** When no critical event decomposition is available, we can resort to techniques that approximate continuous state or action spaces by a finite selection of *samples*.

2 Planning under Prediction Uncertainty

We now begin with algorithms for planning with uncertainty in prediction. Our primary concern here is the need for feedback. Since we cannot plan an explicit sequence of states, we must instead prepare our decision maker for any state it may encounter. Thus we replace the usual action sequences with functions called *policies* that map from state space to action space. To simplify the presentation, we begin with a certain class of degenerate planning problems, namely those in which only a single decision needs to be made. The appropriate extensions to allow multi-stage decision-making (that is, planning) will be made in Section 2.2.

2.1 Making a Single Decision

Let us first consider the problem of making a single decision in the face of uncertainty in the outcome. We will model this uncertainty as decision to be made by another decision maker called *nature*. To formalize, a *single-stage decision problem* is defined by:

- A nonempty action set U that represents the set of choices available to our robot.
- A nonempty parameter set Θ that represents set of choices available to nature. This set should encode all of the uncertainty in the outcome of our agent's decision. In other words, given u and θ, the outcome is fully determined. The value of θ is hidden from the robot.
- A cost (or loss) function $L : U \times \Theta \rightarrow \mathbb{R}$ encoding the relative undesirability of each possible outcome. This is the quantity we will want to minimize. Equivalently, we may define a *reward function* we attempt to maximize.
- An uncertainty model for Θ. Under probabilistic uncertainty, this is the distribution $P(\theta)$. Under nondeterministic uncertainty, we need only a set of possibilities for θ. We may assume that any $\theta \in \Theta$ is allowed, hence, no additional information needs to be specified. (Nondeterministic uncertainty will not be so simple for later formulations.)

The objective is to choose a u that will result in the smallest possible $L(u, \theta)$. However, the outcome of any particular trial is unpredictable. Instead, we'll use the uncertainty model for θ to describe an anticipated outcome. Under nondeterministic uncertainty, the best we can do is to consider *worst case cost*. The worst case optimal decision u^* is

$$u^* = \operatorname*{argmin}_{u \in U} \max_{\theta \in \Theta} L(u, \theta). \tag{1}$$

With the probabilistic uncertainty model, the choice of θ is random, so the relevant measure is the *expected cost*. The decision u^* that minimizes expected cost is

$$u^* = \operatorname*{argmin}_{u \in U} E_\theta[L(u, \theta)] \tag{2}$$

$$= \operatorname*{argmin}_{u \in U} \sum_{\theta \in \Theta} P(\theta) L(u, \theta). \tag{3}$$

In either case, a plan is simply a choice of some $u \in U$ and the problem can be solved with ordinary optimization techniques.

2.1.1 Including an Observation

The previous formulation gave the decision maker no special information about what selection would be made for θ on a particular trial. We may extend the model by including an *observation space* Y. Each $y \in Y$ will correspond to a measurement or reading that we can think of as giving the decision maker a "hint" about the θ that will be selected. The decision maker is given some $y \in Y$ and can use this value when selecting a $u \in U$. Thus, a plan is a *decision rule* (or *strategy* or *policy*) $\gamma : Y \to U$. The presence of observations will change our uncertainty models to be conditioned on the value of y:

- *Nondeterministic:* We assume that y restricts the set of choices available for θ. This can be expressed as a function $F : Y \to 2^\Theta$ so that $F(y) \subseteq \Theta$ represents possible choices for θ given y. Now the optimal decision rule γ^* is simply the one that makes the best worst-case decision for each y:

$$\gamma^*(y) = \operatorname*{argmin}_{u \in U} \max_{\theta \in F(y)} L(u, \theta). \tag{4}$$

  Notice that the only change from (1) is that the max operation is over only $F(y)$, rather than all of Θ

as before.

- *Probabilistic:* The distribution for θ is now conditioned on y. That is, for each $y \in Y$ and $\theta \in \Theta$, we assume that the conditional probability $P(\theta|y)$ is known.

  Given y and u, we can write the expected cost (also called *conditional Bayes risk* in this context) as

$$E^{\theta}[L(u,\theta)] = \sum_{\theta \in \Theta} P(\theta|y)L(u,\theta). \tag{5}$$

  The decision rule to minimize this is

$$\gamma^*(y) = \operatorname*{argmin}_{u \in U} \sum_{\theta \in \Theta} P(\theta|y)L(u,\theta). \tag{6}$$

Two prominent examples of single-stage decision-making with observations are *parameter estimation* [4,5] and *classification* [6–8]. In both, we have $U = \Theta$ and $L(x,\theta) = 0$ if and only if $u = \theta$. The observation y will give some information about θ, perhaps as a feature vector or a noise-tainted estimate of θ.

2.1.2 Criticisms of Decision Theory

This is an appropriate point to scrutinize the assumptions implicit in the the use of decision-theoretic methods.

Generating cost functions First, most decision theoretic methods depend on a cost function L which must be selected by hand for each problem. Choosing an appropriate cost function may be difficult. *Utility theory* [5,9,10] deals with the existence and, to a lesser degree, construction of these cost functions under the assumption that the decision maker is, in a precisely-defined way, reasonably rational. Note also that some formulations can be reworked to eliminate the need for quantification of costs. For example, the minimax formulation of (1) really only requires a total ordering on $U \times \Theta$, rather than a real-valued cost function, to make sense. More generally, many decision theoretic methods can be augmented with *sensitivity analysis*, which is a way of quantifying the amount of disturbance in L needed to make some change in the optimal policy. The idea is that if the policy is fairly robust to changes in L, then a poorly-crafted cost function will not have much effect on the decisions made.

Pessimism and nondeterministic uncertainty Nondeterministic models for uncertainty are often criticized for being overly pessimistic. In fact, using nondeterministic uncertainty with worst-case analysis can cause serious limitations on the planning problems that can be solved. Section 2.3.1 will highlight this problem in the context of the convergence of value iteration. Of course, the fact that we express uncertainty as a set of possible outcomes does not constrain us to worst case analysis. One can easily imagine an optimistic "best-cast" version of (4):

$$\gamma^*(y) = \operatorname*{argmin}_{u \in U} \min_{\theta \in F(y)} L(u, \theta). \tag{7}$$

This is still unsatisfying because we have simply traded excessive pessimism for an equal measure of optimism. A compromise approach called *Hurwicz weighting* involves selecting a parameter $\alpha \in [0, 1]$ that is in some sense a "coefficient of optimism". We can use α to blend (4) with (7):

$$\gamma^*(y) = \operatorname*{argmin}_{u \in U} \left\{ \alpha \left[\min_{\theta \in F(y)} L(u, \theta) \right] + (1 - \alpha) \left[\max_{\theta \in F(y)} L(u, \theta) \right] \right\}. \tag{8}$$

What is probability? There is also debate about the proper understanding of probabilities. The *Bayesian* interpretation views probability as a belief about a single trial. This is essentially the interpretation we have used so far. Given y, a Bayesian thinks of $P(\theta|y)$ as a degree of belief that nature will select θ. In contrast, the *frequentist* interpretation believes that probability is only properly understood in the limit as the number of trials goes to infinity; a probability value says nothing to a frequentist about the next trial, but only about the limit of an infinite sequence of trials. Frequentist interpretations of probability have led to a different, more conservative form of decision theory [10].

2.2 Making a Sequence of Decisions

In the previous section, we considered the problem of making a single decision in the face of some uncertainty in the outcome. We may think of planning under prediction uncertainty as a generalization of this idea by introducing a state space X and allowing a sequence of successive decisions to influence the system's transitions between states in X.

We divide time into *stages* and number them starting with 1. Both the robot and nature make a decision at each stage. For the moment, suppose that the number of stages is limited to K. We will relax this restriction momentarily. Let $\tilde{u} = (u_1, u_2, \ldots, u_K)$ and $\tilde{\theta} = (\theta_1, \theta_2, \ldots, \theta_K)$ denote the sequences of decisions made by the

robot and nature respectively. Given an initial state x_1, we can define a state sequence $\tilde{x} = (x_1, x_2, \ldots, x_{K+1})$ according to a deterministic transition function: $x_{k+1} = f(x_k, u_k, \theta_k)$. Fig. 1 summarizes this situation for a single stage. To state the problem more formally, we need:

- A nonempty state set X.
- A nonempty action set U. Alternatively, the set of available actions may depend on the current state, i.e. we have a set $U(x)$ for each $x \in X$. Since this variation only clutters the notation without making the problem more interesting, we assume that the same actions are available from each state. One possible realization of this action set is that lower-level techniques like motion planning, map building, and manipulation are are implementations of the abstract actions we consider. This sort of layered approach has been used in a number of successful robotic systems [11–16].
- A nature action set Θ. As with U, nature's available actions may depend on x.
- A deterministic state transition function $f : X \times U \times \Theta \to X$.
- An initial state x_1.
- A *stage-additive* cost functional

$$L(\tilde{x}, \tilde{u}, \tilde{\theta}) = \sum_{k=1}^{K} l(x_k, u_k, \theta_k) + l_F(x_{K+1}). \tag{9}$$

  The cost functional L is defined in terms of *single-stage cost function* $l : X \times U \times \theta \to \mathbb{R} \cup \{\infty\}$ that gives the cost for each possible transition, and a *termination cost function* $l_F : X \to \mathbb{R} \cup \{\infty\}$ that gives a cost for being in each state when execution ends after K stages. Sometimes it will be convenient to discuss the special case where the single-stage cost depends only on x_k and u_k. In such cases we write simply $l(x_k, u_k)$.
- A goal region X_G. For each $x_g \in X_G$, we require $l_F(x_g) = 0$.
- An uncertainty model for Θ. As usual, we allow either probabilistic or nondeterministic uncertainty. For nondeterministic uncertainty we need for each x and u a set of possibilities $\Theta(x, u)$. In the probabilistic case, we need a distribution $P(\theta|x, u)$.

Feasible Planning As an example, suppose we are not interested in optimizing any cost measure but only in reaching X_G. To accomplish this, we can set $l(x, u) = 0$ for all $x \in X$ and $u \in U$ and set

$$l_F(x) = \begin{cases} 0 & \text{if } x \in X_G \\ \infty & \text{otherwise} \end{cases}. \tag{10}$$

With this cost functional, any plan execution that terminates in X_G will have cost 0; any execution that terminates outside X_G will have infinite cost.

Allowing executions of indefinite length Now we relax the assumption that our decision maker will act for a predetermined number of stages. Introduce into U a fictitious *termination action* u_F which indicates the decision maker's intention to end the execution. Create a fictitious state x_F, to which selecting u_F always leads. Select $U(x_F) = \{u_F\}$ so that once the agent has terminated, it cannot restart. Lastly, assign $l(x_F, u_F, \theta) = 0$ for all θ.

Now we imagine that stages continue infinitely, so that $\tilde{x}$, $\tilde{u}$ and $\tilde{\theta}$ become infinite sequences and the accumulated cost for an execution is

$$L(\tilde{x}, \tilde{u}, \tilde{\theta}) = \sum_{k=1}^{\infty} l(x_k, u_k, \theta_k). \tag{11}$$

Now we can *define* K in terms of the actions selected, instead of assuming it is known ahead of time:

$$K = \min\{k | u_k = u_F\}. \tag{12}$$

If the robot eventually selects u_F, then K is well-defined. We neglect cases in which the robot never chooses u_F because the cost of such an execution will generally increase without bound. By defining $l_F(x) = l(x, u_F, \theta)$ for all x and θ, we ensure that (9) still holds.

Defining an optimal policy A solution to this type of problem is a *policy* $\gamma : X \to U$ that produces an action for each state. In the sequel, the terms plan and policy are interchangeable.

Consider nondeterministic uncertainty. Just as we did in the single-stage case, we want to select a policy that minimizes the worst-case cost. For a single decision, that maximization was over nature's choices for

θ. Now the cost of a single execution of a plan depends on the entire sequence of choices made by both the robot and nature, namely $\tilde{u}$ and $\tilde{\theta}$, as well as $\tilde{x}$, which they determine. For a policy π, let $\mathcal{H}(\pi, x_1)$ denote the set of all such histories that can result from executing π starting at x_1. This is the set over which we must consider the worst case. Let $G_\pi(x_1)$ denote the worst-case cost of executing the policy π starting from state x_1:

$$G_\pi(x_1) = \max_{(\tilde{x},\tilde{u},\tilde{\theta})\in\mathcal{H}(\pi,x_1)} L(\tilde{x},\tilde{u},\tilde{\theta}). \tag{13}$$

The probabilistic case is similar, using expectation instead of instead of worst-case analysis:

$$G_\pi(x_1) = E_{\mathcal{H}(\pi,x_1)}\left[L(\tilde{x},\tilde{u},\tilde{\theta})\right]. \tag{14}$$

For either sort of uncertainty, an *optimal policy* π^* is one that minimizes G:

$$\pi^* = \operatorname*{argmin}_\pi G_\pi(x_1), \tag{15}$$

where the minimum is over all possible policies. Some readers may have noticed that this definition depends on the initial state x_1. Fortunately, there will exist a single policy that is optimal regardless of initial state. Suppose a policy π^* achieves the minimum in (15) for a fixed x_1 and let x denote a state reachable from x_1. If π^* were not optimal from x, then the goal could be reached from x_1 via x with lower cost than by executing π^*, contradicting the optimality of π^*. Consequently there will exist a single policy that is optimal regardless of initial state.

2.3 Methods for Finding Optimal Solutions

We have defined a general type of planning problem that includes uncertainty in state transitions and defined a notion of a solution to such a problem. Now we turn our attention to general-purpose solution methods for these problems. As one might expect, we must carefully weigh the trade-offs between generality, optimality, and tractability. Since optimality will come only at a high computational cost, we consider approximate solution methods in Section 2.4. In one sense, computing an optimal plan is just an optimization problem over the extremely large space of all policies. Fortunately, our optimality criterion G exhibits enough structure to make several different kinds dynamic programming possible.

2.3.1 Value Iteration

Value iteration [17] is so named because it gradually develops a *value function* or *cost-to-go function* from which an optimal policy can be extracted. We will derive a recursive expression for this value function; this recurrence will lead directly to a planning algorithm. The derivation proceeds slightly differently depending on the uncertainty model.

Nondeterministic uncertainty Fix a stage k and let $G_k^*(x_k)$ denote the worst-case cost that could accumulate if the robot executes π^* starting at x_k. We can write $G_k^*(x_k)$ as an alternation of minimum (from the optimality of π^*) and maximum (from the use of worst-case analysis) operations:

$$G_k^*(x_k) = \min_{u_k} \max_{\theta_k} \min_{u_{k+1}} \max_{\theta_{k+1}} \cdots \min_{u_K} \max_{\theta_K} \left\{ \sum_{i=k}^{K} l(x_i, u_i, \theta_i) + l_F(x_{K+1}) \right\}. \tag{16}$$

Suppose we separate the first term $l(x_k, u_k, \theta_k)$ from the summation. Since this term only affects the outermost min and max operations, we can extract it from all of the others to get

$$G_k^*(x_k) = \min_{u_k} \max_{\theta_k} \left\{ l(x_k, u_k, \theta_k) + \min_{u_{k+1}} \max_{\theta_{k+1}} \cdots \min_{u_K} \max_{\theta_K} \left[\sum_{i=k+1}^{K} l(x_i, u_i, \theta_i) + l_F(x_{K+1}) \right] \right\}. \tag{17}$$

Notice that the innermost portion of (17) is simply $G_{k+1}(x_{k+1})$, leaving a simple recurrence:

$$G_k^*(x_k) = \min_{u_k} \max_{\theta_k} \left\{ l(x_k, u_k, \theta_k) + G_{k+1}^*(x_{k+1}) \right\}. \tag{18}$$

We also have a simple base case:

$$G_{K+1}^*(x_{K+1}) = l_F(x_F). \tag{19}$$

The value iteration algorithm is a direct implementation of this recurrence. In iteration i of the algorithm, we use the values of G_{K-i+1}^* from the previous iteration (or, when $i = 0$, from the base case) to compute G_{K-i}^* according to (18). Of course, K, the number of actions taken by the robot before terminating, is not known ahead of time. One way to think of this is that the algorithm starts with the stage in which the robot terminates and move backward in time, considering progressively longer executions that lead to termination. The value of K never becomes relevant to the execution of the algorithm.

An implementation might be based on two tables, each with one entry for each state. At iteration i, one table holds the values of G^*_{K-i+1} while the other is filled in with G^*_{K-i}. After an iteration finishes, the roles of these tables can be swapped in preparation for the next iteration.

We want to terminate the value iteration algorithm when we reach an iteration in which no change occurs, that is, when an iteration i is reached in which $G^*_{K-i} = G^*_{K-i+1}$. If this occurs, then we will have reached a *stationary value function* $G^* = G^*_{K-i}$ that gives the worst-case cost that will result from executing an optimal policy starting from each state. This convergence will occur for all states from which there exists some policy that can guarantee reaching X_G. If no policy can guarantee reaching X_G, then no stationary value function exists and value iteration will not converge. Fig. 2 shows a simple example of each case.

Finally, given a stationary value function G^*, we can extract an optimal policy π^* in a straightforward way. When the robot is in state x_k, we want to choose the u_k that achieves the minimum in (18):

$$\pi^*(x) = \operatorname*{argmin}_u \max_\theta \{l(x, u, \theta) + G^*(f(x, u, \theta)\} . \tag{20}$$

Probabilistic uncertainty Under probabilistic uncertainty, a very similar approach will work, because of the linearity of expectation:

$$G^*_k(x_k) = \min_{u_k,\ldots,u_K} E_{\theta_k,\ldots,\theta_K} \left[\sum_{i=k}^{K} l(x_i, u_i, \theta_i) + l_F(x_{K+1}) \right] \tag{21}$$

$$= \min_{u_k} E_{\theta_k} \left[l(x_k, u_k, \theta_k) + G^*_{k+1}(x_{k+1}) \right] . \tag{22}$$

The base case is the same:

$$G^*_{K+1}(x_{K+1}) = l_F(x_F) \tag{23}$$

Equations (19) and (22) provide the base case and recursive case for value iteration, which works in just the same way as in the nondeterministic case. Convergence, however, is an even thornier question than in the nondeterministic case, because of the possibility that the costs-to-go will converge only in the limit. Fig. 3 shows an extremely simple example in which this is the case. This phenomenon will occur any time there is nonzero probability of being forced by nature to traverse cycles in the state space. For many applications, the costs-to-go will converge quickly to good approximations of the optimal values. More importantly, recall that we are not directly interested in G^*, but in the policy π^* we extract from it. Thus, we only need the

cost-to-go to converge to a point where we are reasonably certain of which action is the correct choice from each state.

Finally, when the dynamic programming iterations finish, we can use the resulting G^* to extract an optimal policy. The probabilistic analog to (20) is

$$\pi^*(x) = \operatorname*{argmin}_u E_\theta[l(x,u,\theta) + G^*(f(x,u,\theta)]. \tag{24}$$

2.3.2 Policy Iteration

Value iteration was a dynamic programming technique in the space of states. Only after the stationary cost-to-go function (or an approximation of it) is reached can a policy be extracted. In contrast, *policy iteration* [17, 18] performs dynamic programming directly in the space of policies. At each iteration, a fully-formed policy is generated.

Each step of policy iteration has two parts: *policy evaluation*, in which the expected cost of executing the current policy is computed and *policy improvement*, in which this information is used to construct a policy better than the current one. To simplify notation, assume that the cost of each transition depends on only x and u, so that we can write $l(x,u)$ rather than $l(x,u,\theta)$.

Policy evaluation First, how can we evaluate a fixed policy π? Recall that $G_\pi(x)$ denotes the expected cost of executing π starting at x. The values of $G_\pi(x)$ will serve as our criteria for evaluating π. We can derive an expression for $G_\pi(x)$ in a similar manner to the derivation of (22), but in which we restrict the available actions in each state to the single action suggested by π:

$$\begin{aligned} G_\pi(x) &= E_\theta\left[l(x,\pi(x)) + G_\pi(f(x,u,\theta))\right] &(25)\\ &= l(x,\pi(x)) + \sum_{x' \in X} G_\pi(x')P(x'|x,u). &(26)\end{aligned}$$

The transition probability $P(x'|x,u)$ can be obtained by marginalizing over θ:

$$P(x'|x,u) = \sum_{\{\theta | f(x,u,\theta)=x'\}} P(\theta|x,u). \tag{27}$$

Define $n = |X|$. Equation (26) is a linear equation with n unknowns, namely $G_\pi(x)$ for each x. If we

make n copies of (26), one for each $x \in X$, we get a linear system with n variables and n equations. Solving this system with standard linear algebra methods (singular value decomposition [19], for example) gives values for $G_\pi(x)$.

Policy improvement Now we will show how to use G_π to generate a new policy π' that is an improvement over π in the sense that $G_{\pi'}(x) \leq G_\pi(x)$ for all x. We can construct π' in a relatively direct way. For each x define $\pi'(x)$ according to

$$\pi'(x) = \underset{u \in U}{\operatorname{argmin}} \left\{ l(x,u) + \sum_{x' \in X} G_\pi(x') P(x'|x,u) \right\}. \tag{28}$$

This is probably best understood in relation to (24). The real difference is that during execution of policy iteration, G^* is unknown. Instead, we use G_π as an estimate for G^*. Since π' will take the best action from x under the assumption that $G_\pi(x')$ is the cost-to-go after this step, we can conclude that π' is at least as good as π. If $\pi' = \pi$, then the algorithm has converged to π^* and can terminate.

One important property of policy iteration is that, unlike value iteration, it is guaranteed to terminate in finite time. Since an improvement is made on each iteration, no policy will occur more than once. But there are only $|U|^{|X|}$ different policies to consider. Therefore, this algorithm will terminate with the optimal policy in at most $|U|^{|X|}$ iterations, but generally much faster than this.

2.3.3 Other Methods

We have focused on only two optimal algorithms in order to provide some amount of depth to the subject, and because many other algorithms can be seen as variants of either policy iteration or value iteration. The versions we describe are a form of backward dynamic programming in the sense that they begin with termination and progress backward in time. Forward versions are also possible [17], but slightly more complex conceptually. We described value iteration as a series of sweeps across the state space performing updates but the ordering of updates allows more flexibility. In some cases this property can be exploited to find good policies faster using so-called *asynchronous methods* [20–22]. Under certain restrictions, Dijkstra's shortest-path algorithm can be adapted to account for uncertainty [23].

2.4 Methods for Finding Approximate Solutions

Now let us turn our attention to algorithms for planning that is only approximately optimal. Suboptimal planning is important for problems that are too complex to solve optimally and for situations in which resources for computation are limited. For example, if an autonomous robot suddenly discovers an error in its model of the world (say, an unexpected obstacle in its path), it must quickly *replan* under its new world model. In such a circumstance, a plan must be generated quickly and computing an optimal plan may not be possible. We have already seen one suboptimal planning algorithm – the prematurely terminated version of value iteration that arose for probabilistic problems that converge only in the limit. There are also a number of more specialized algorithms.

2.4.1 Certainty Equivalent Control

Allowing even only prediction uncertainty makes planning much more difficult. What happens if we ignore the uncertainty when generating a plan? This is the idea behind *certainty equivalent control*. More precisely, we create an uncertainty-free planning problem by assuming that uncertain parameters will take on "typical" values. So in our formulation, we might form a deterministic planning problem by defining a deterministic state transition function $\bar{f}$ according to the most likely successor:

$$\bar{f}(x_k, u_k) = f(x_k, u_k, \arg\max_{\theta_k} P(\theta_k | x_k, u_k)). \tag{29}$$

In the special case where states are numbers, a "typical" result might be the expected one:

$$\bar{f}(x_k, u_k) = f(x_k, u_k, E_{\theta_k}[x_k]). \tag{30}$$

By solving the planning problem with transition function $\bar{f}$, we get a plan for the original problem under f. Remarkably, for a certain classes of systems (eg. linear systems with quadratic cost), this method has been shown to generate optimal plans [24].

2.4.2 Limited Lookahead

Limited lookahead (or *rolling horizon approximation*) is an approximation technique that aims to reduce the computation required in value iteration. Suppose we have some estimate of the optimal cost-to-go $\hat{G} \approx G^*$.

We use this as the base case for value iteration, replacing (19). (Some readers may recognize this as essentially the same method that drives computer game-playing, in which $\hat{G}$ is called an *evaluation function*.) If we run i rounds of value iteration with $\hat{G}$ as the base case, the resulting policy will be optimal for the simplified problem in which the decision maker acts for i stages before terminating with cost $\hat{G}(x_k)$. The similarity of this policy to π^* depends directly on the similarity of $\hat{G}$ to G^*.

One question that remains is how to select $\hat{G}$. One possibility is to use some heuristic method to generate a *base policy* $\hat{\pi}$ and use its cost-to-go as $\hat{G}$:

$$\hat{G} = G_{\hat{\pi}}. \tag{31}$$

One-step lookahead algorithms built on a base policy in this way are called *rollout algorithms*. This rollout can be viewed as a single step of policy iteration in the sense that the cost-to-go function of one policy, $\hat{\pi}$, is used to create a new, improved policy.

2.5 Conquering Continuous Spaces

Until now, we have talked about methods which handle problems with finite state spaces and finite action sets. In many situations, especially in robotics, continuous state spaces and action sets are more natural. By continuous, we mean that either the state space, the action set, or both have an uncountably infinite number of elements. In this section, we will extend the above methods to problems with continuous spaces. The main difficulty is that the techniques we have presented depend on iterating over the elements of X and U. The key idea of the extension is to find a suitable finite representation of the original problem. Then the new problem can be solved with methods similar to those we have already developed.

The first step of the transformation process is to approximate the continuous state space with a finite sampling point set. These sampling points could be obtained by any of several methods, such as random sampling [25], quasi-random sampling [26], grid sampling [27], or lattice point sampling [28–30]. In selecting one of these sampling methods, one must consider several issues, including the *uniformity* of the points (How well are the points spread out?) and *neighborhood structure* (Given a point in the underlying space, how easy it it to locate its neighbors in the sample set?). Some example sets of sampling points in the unit square are shown in Figure 4. A more thorough characterization and comparison of sampling techniques appears in Section 5.2 of [23].

After the continuous state space is represented by a finite set, standard value iteration as described in

Section 2.3.1 can be applied with the following modification. As the algorithm proceeds, we maintain the value function $G_k^*(x_k)$ only for states in the sampling set. Recall that the update equation (either 18 or 22) depends on knowing $G_{k+1}^*(x_{k+1})$ for each choice of u_k and θ_k [1]. If x_{k+1} is not in the sample set, then $G_{k+1}^*(x_{k+1})$ will not be available, as illustrated in Figure 5.

To make value iteration work, the value function at x_{k+1} needs to be approximated with values of states in the sampled set [31]. In [32, 33], a neural network is used to approximate the value function. A more conventional way is by *interpolation*. Interpolation involves designating some set of samples as "neighbors" of x_{k+1} and using as the value function at x_{k+1} some weighted combination of the value function at each of these neighbors. Interpolation has been most thoroughly studied for the case in which the sample set is a grid. In this case, the interpolation can be performed in at least two different ways:

- **Multi-linear interpolation.** In the one-dimensional case, multi-linear interpolation is just linear interpolation between data points. In higher dimensions, the procedure is recursive:

  1. Choose any axis and project the point onto two faces that are perpendicular to the chosen axis.
  2. Use $(n-1)$-dimensional multi-linear interpolation to calculate the value function at these two points.
  3. Linearly interpolate to calculate the value of the given point according to the value of two points on the two faces.

  Multi-linear interpolation will process 2^n data points for one interpolation in n-dimensional state space. It can very time-consuming for high-dimensional problems.

- **Simplex-based interpolation [34–37].** This method uses Kuhn triangulation to decompose the n-dimensional hypercube into $n!$ simplices, each of which has $n+1$ vertices. Then the simplex-based interpolation is to calculate the value according to the $n+1$ vertices of the simplex containing the given point. Since only $n+1$ data points and $O(n \log n)$ time will be needed in one interpolation, it is much more efficient than multi-linear interpolation method.

Finally, with the finite sampled set and a chosen interpolation method, we could run value iteration as in the discrete case, but interpolating to estimate the value function for states that are not in the sample set.

[1] If U and Θ are themselves continuous, it may be necessary to sample them as well.

Similarly, the optimal policy can be extracted from either (20) or (24), again using interpolation to estimate G^*. More details on sampling-based dynamic programming in continuous state spaces appear in [17,38–41].

2.6 Variations

We conclude this section with a survey of variations to the problem formulated in Section 2.2.

2.6.1 Infinite Horizon Models

The model presented in Section 2.2 deals with planning problems with a well-defined goal set. The decision-maker interacts with the environment for a finite period of time before selecting a termination action. What happens if we eliminate the termination actions and allow the robot to continue executing for an infinite number of stages? Problems of this type are called *infinite horizon* problems and have been studied extensively in artificial intelligence and stochastic control theory.

Since the process is infinite, we can omit x_1, X_G and l_F from the model. However, the most striking change is in the cost functional $L(\tilde{x}, \tilde{u}, \tilde{\theta})$. Excluding the case in which there are cycles with zero or negative cost in which the robot can linger, allowing K to approach infinity in (9) will cause L to diverge. As a result, (9) is no longer a suitable optimality criterion. We must find a way of keeping the cost finite for an infinite sequence of actions. Two possibilities are:

- **Average Cost Per Stage** (or *gain-optimal* cost): One way to keep the cost finite is to divide by the number of stages:

$$L(\tilde{x}, \tilde{u}, \tilde{\theta}) = \lim_{K \to \infty} \frac{1}{K} \sum_{k=1}^{K} l(x_k, u_k, \theta_k). \tag{32}$$

  If l is bounded by some constant, then it is clear that L must remain less than this constant. One major problem with this model is that costs over any initial prefix are overshadowed by long-run performance [42].

- **Discounted Cost:** Pick a parameter $\alpha \in (0, 1)$ called a *discount factor* and use α to define L in the following way:

$$L(\tilde{x}, \tilde{u}, \tilde{\theta}) = \sum_{k=1}^{\infty} \alpha^{k-1} l(x_k, u_k, \theta_k). \tag{33}$$

  The intuition is to place less weight on costs that occur further into the future. We may think of α

as a measure of "far-sightedness". If α is increased, loss from later stages has greater influence on the value of L. The average cost model can be seen as a limiting case as α approaches 1 [43].

It is important to understand that α is a part of the definition of the optimal policy. A change in α can result in a change in which actions are considered optimal from each state. For this reason, the average cost model is sometimes preferred because it does not introduce any new parameters to tune. Regardless, discounted cost is the dominant model because it is a simple and mathematically manageable way to keep finite the cost of an infinite length execution.

The dynamic programming methods of Sections 2.3 can be adapted to find optimal policies for discounted costs, but care must be taken to ensure the stability and convergence properties of these algorithms. A detailed treatment of these methods and others for infinite horizon problems is in [43]. In Section 2.6.2, we discuss reinforcement learning, which generally uses a discounted cost model, but assumes that the uncertainty model for θ is unknown.

2.6.2 Reinforcement Learning

One of the major problems with decision-theoretic planning as we have presented it is that there is a heavy modeling burden associated with the assumption that an uncertainty model for Θ is given. In the probabilistic case, this is the assumption that the distribution $P(x'|x, u)$ is known. When X and U are finite, there are still $|X|\,|U|$ separate values needed to describe this distribution. In a physical system, each of these would require many trials to estimate accurately. For many nontrivial environments, this can be quite impractical.

An alternative is to force the robot to learn these probabilities along the way instead of specifying them up front. The family of methods that takes this approach is generally called *reinforcement learning* (RL), and occasionally *neuro-dynamic programming* [32] (although that term has a somewhat more specific meaning) or *simulation-based methods* [17]. The primary difference from methods that assume that transition probabilities are known is that reinforcement learning is an *online* model. This means that there is no separation between planning and execution. Rather, as the robot interacts with the environment, it gradually refines its plan. We will very briefly describe an algorithm that, for reasons that will soon be obvious, is called *Q-learning* [44]. Q-learning is quite simple, but worth understanding because nearly all other RL algorithms can be seen as variations on the same basic themes.

We can think of Q-learning as type of value iteration in which, instead of using G^* (which gives the value

of each state), uses a function $Q : X \times U \to \mathbb{R}$ that gives values for state-action pairs. More precisely, we define $Q(x, u)$ to be the expected cost of starting from state x, taking action u, and acting optimally (that is, according to π^*) thereafter.

The algorithm works by maintaining a table that lists, for each state-action pair, an estimate $\hat{Q}(x, u)$ of the real $Q(x, u)$. We initialize $\hat{Q}(x, u)$ arbitrarily. After each action, the agent is informed of the new state x' and the cost l of the corresponding transition and an update to the table is performed:

$$\hat{Q}(x, u) \leftarrow (1 - \rho)\hat{Q}(x, u) + \rho \left(l + \min_{u' \in U(x')} \hat{Q}(x', u') \right). \tag{34}$$

If we use the discounted-cost infinite horizon model (as is the custom in the reinforcement learning literature), we must include the discount factor:

$$\hat{Q}(x, u) \leftarrow (1 - \rho)\hat{Q}(x, u) + \rho \left(l + \alpha \min_{u' \in U(x')} \hat{Q}(x', u') \right). \tag{35}$$

In either update rule ρ is a designer-specified *convergence rate* or *learning rate*. It has been shown (for example, in [8]) that with certain assumptions about the sequence of actions chosen, $\hat{Q}$ will converge to Q under this update rule. It may seem conspicuous that the update rule never mentions the transition probability $P(x'|x, u)$. In fact, Q-learning is an example of so-called *model-free* algorithms that never build an explicit model of the transition probabilities. Instead, the probabilities are hidden by the fact that the update in (34) or (35) is performed repeatedly, with the distribution of resulting states chosen according to $P(x'|x, u)$. Thus, successor states that are more likely will have greater influence over $\hat{Q}(x, u)$.

Exploration vs. exploitation To this point, we've shown how Q-learning maintains an estimate for the value of each state-action pair without saying anything about which actions to choose. Suppose $Q(x, u)$ is known for all x and u. Then the best action to choose (cf. (20)) is

$$\pi^*(x) = \operatorname*{argmin}_u Q(x, u). \tag{36}$$

Unfortunately, the decision maker must choose actions without knowledge of the real Q. To obtain the best cost over a limited period of time, there is a tension between *exploring* the space in order to make $\hat{Q}$ a better estimate of Q and *exploiting* actions that have been effective so far – that is, actions for which $\hat{Q}$ is currently

small.

The exploration-exploitation dilemma is an important enough problem in reinforcement learning that many different methods have been suggested to deal with it, including *initial optimism* [45], which assigns a large initial values to each $\hat{Q}(x,u)$, ensuring that each action is tried often enough to "drive down" its $\hat{Q}$ value to near its true value, and ϵ-greedy policies [33] that select the current best action with probability ϵ and choose randomly otherwise. A in-depth study of this topic is in [46].

Temporal credit assignment While the update rule (34) is guaranteed to converge, in practice this can require a large number of trials to reach a good estimate for Q. The issue is that of *credit assignment*: When a reward is received, to which actions to we attribute it? In (34), credit is assigned only to the action immediately preceding the reward. This is troubling because if a large reward (say, obtaining a Ph.D.) occurs, credit for this reward will initially only be granted to the action that immediately led to this reward (finishing a dissertation) and not to any earlier actions that made the reward possible (enrolling in graduate school). Only after earning many Ph.D.s (!) will the influence of this reward propagate backward to have an influence on the decision to enroll in graduate school.

To combat this problem, more aggressive credit-assignment schemes have been developed that endeavor to squeeze more out of each action taken by the decision maker. One of the most effective techniques is to maintain an *eligibility trace* – a list of recent state-action pairs along with a weight for each. Eligibility traces are so named because they determine which $\hat{Q}(x,u)$ values are eligible to be updated after the next action. Recently visited states are marked as eligible. After receiving a new cost l, we perform an update similar to (34) for each eligible state-action pair. As time passes, the weight of each eligible state decays (reducing the amount of change in its $\hat{Q}$ on subsequent iterations) until it is finally removed from the eligible list.

2.6.3 Additional Decision Makers

Everything up to this point has focused on a single decision maker interacting with an uncertain environment. Uncertainty is modeled as a "game against nature". A broad class of generalizations can be made if we allow additional decision makers in the system, each with its own independent goals. This is the realm of *game theory* [47,48], which is concerned with the general situation of multiple decision makers interacting in some way. The breadth and depth of game theory literature will force us to focus on just a few issues that are vital for planning in the presence of other decision-makers.

Suppose we have n decision makers (not counting nature). At each stage, decision maker i will select an action from an action set U^i that has some influence on the resulting state. This means we must extend the state transition function:

$$f : X \times U^1 \times \cdots \times U^n \times \Theta \to X. \tag{37}$$

Each decision maker also has its own cost functional L^i which depends on all n actions selected at each stage. It is assumed that each decision maker has complete knowledge of all of the L^i's. The special case where $n = 2$ and $L^1(\tilde{x}, \tilde{u}, \tilde{\theta}) + L^2(\tilde{x}, \tilde{u}, \tilde{\theta}) = 0$ is unsurprisingly called a *zero-sum game*. This corresponds to the situation where two players are in direct competition for some limited resource.

What does a plan look like when there are multiple decision makers? The deterministic policies considered so far are no longer adequate. Fig. 6 illustrates a very simple problem that requires a *mixed strategy* that selects actions at random according to some distribution. By contrast, the deterministic strategies we studied for the single-agent case are also called *pure strategies*.

With a single decision-maker, we defined optimality in terms of the expected or worst-case cost. When there are multiple decision makers, optimality is usually defined in terms of *regret*, which is a measure of how much a decision-maker could have improved his reward if he had known what actions the other players would take. For a single stage game in which the actions selected are $u^1, \ldots, u^n$, the regret R^i for player i is

$$R^i = L^i(u^1, \ldots, u^n) - \max_{u' \in U^i} L^i(u^1, \ldots, u', \ldots, u^n). \tag{38}$$

For two-player zero-sum games, a pair of policies for which $R^1 = R^2 = 0$ is called a *saddle point*. A fundamental result in game theory is that if mixed strategies are allowed, then a saddle point will always exist. For nonzero-sum games and those with multiple players, the idea of a saddle points can be generalized to *Nash equilibria*, which are also based on the idea of eliminating regret.

3 Planning under Sensing Uncertainty

In this section we address the planning problem in which the knowledge of the robot's current state is limited, or not available at all. This accounts for the cases when the robot's sensors do not uniquely determine the current state of the robot (sensing uncertainty) and when the robot's control is not perfect (prediction

uncertainty).

One common approach is to make an estimation of the current state, with all the information available, and determine some bound for the state uncertainty. Then the uncertainty may be *ignored*, and the algorithms of the previous sections may be applied. However, the state estimation may be completely avoided in the computation of a plan, i.e., the robot may be able to reach achieve its goal without ever determining its current state. This gives rise to the study of the *information space*, which will be the main topic of this section.

Information spaces have appeared throughout the robotics literature in many forms and under many different names. Information space concepts arise in maze searching [49], preimage planning [50], error detection and recovery [51], manipulation [52–55], bug algorithms [56, 57], gap navigation trees [58, 59], perceptual kinematic maps [60], perceptual equivalence classes and information invariants [61, 62], sensor-based planning [63], searching unknown dynamic environments, D^* [64], pursuit-evasion [65–69], probabilistic navigation [70], Bayesian localization and SLAM [71,72], and guaranteed localization [73–75], and topological maps [76], to cite just a few examples.

In general, the robot can gather information about the state from the following sources:

- **Initial Conditions:** Information the robot has about the task before the first sensing measurement is taken or the first action is performed. The particular initial condition for a planning problem, denoted by η_0, can have several forms:
  - **Known State:** The initial state $x_1 \in X$ is given. Uncertainty appears when nature interferes with the state transition equation.
  - **Nondeterministic:** A set $X_1 \subset X$ is given. The initial state is known to lie within X_1.
  - **Probabilistic:** A probability distribution $P(x_1)$ over X is given.
- **Sensor observations:** Online measurements of the state are made. In general they do not give all the information of the state, either because some state variable cannot be measured (a sensor for it is not available), or due to limitations in the sensor construction, sensor resolution, disturbances due to noise, etc.
- **Previous actions:** The record of the actions may provide the robot with useful information. For example, under the assumption of perfect control, if the previous action was to move to the east, the

current state is more to the east as the previous state, although neither the previous nor the current state is known.

- **Available actions:** The state may be inferred from knowledge of what actions are available to the robot.

3.1 Discrete State Spaces

We first describe the information space when X, the state space, is finite or countably infinite. The new element for computing a plan is that the robot does not have a complete knowledge of the current state, but it can measure it in some way through observations. Because of this, we begin our discussions with modeling the robot's *sensors*.

3.1.1 Sensors

A sensor is a device that provides some measurement of the current state. When the robot performs a sensing in the environment, the sensor *maps* the state space into the observation space Y. The observation space is the set of all possible readings of the sensor, giving 'hints' of the current state. This is different from the case presented in Section 2.1.1, in which the observation only gave hints of the possible action that nature would take. The sensor mapping, denoted by h, takes several forms:

- **State sensor mapping:** Given a state $x \in X$, the observation $y = h(x) \in Y$ is completely determined.
- **State-nature sensor mapping:** Nature is allowed to interfere with the sensor measurements. Let $\Psi(x)$ denote a finite set of *nature sensing actions*, defined for each $x \in X$. The mapping produces an observation $y = h(x, \psi)$ for every $x \in X$ and for every $\psi \in \Psi(x)$. As with Θ in Section 2.1, the particular ψ chosen by nature is assumed to be unknown.
- **History based sensor mapping:** This case is similar to the last one, but the observation may depend on previous states. If the plan is in stage k, the observation is $y = h_k(x_1, x_2, \ldots, x_k, \psi_k)$. In this case $\psi_k \in \Psi_k$ is the particular sensing action chosen by nature.

Adding sensing uncertainty to the model of Figure 1, yields the model presented in Figure 7. The decision maker does not have direct access to the state, which can only be measured through sensors.

3.1.2 Definition of the Information Space

Let X, U and f follow the same definitions as in Section 2.2. If the plan is at stage k, we want to determine which information is available to the robot, either from the new observations, or the accumulation of previous information. It is assumed that the robot keeps a record of each of the observations made. Thus, the *observation history*, $\tilde{y} = (y_1, y_2, \ldots, y_k)$, is the ordered sequence of observations up to state k. Similarly, the *action history*, $\tilde{u} = (u_1, u_2, \ldots, u_{k-1})$, is the record of the actions taken. It runs until stage $k-1$, because action u_{k-1} is applied in state x_{k-1}, to yield the current state x_k, where the observation y_k is made. Remember that η_0 denotes the initial condition. The *information state* at state k is defined as

$$\eta_k = (\eta_0, \tilde{u}_{k-1}, \tilde{y}_k), \tag{39}$$

that is, the initial condition together with the history. Alternatively, an information state can be expressed recursively as

$$\eta_k = (\eta_{k-1}, u_{k-1}, y_k), \tag{40}$$

since the difference between the previous and the current information state consists of the new observation made and the new action taken.

The set of all possible information states η_i for $1 \leq i \leq k$, is called the *information space*, $\mathcal{I}$. Similar to the case of prediction uncertainty, presented in Section 2.2, a plan in the information state is defined as a mapping π, but in this case using the information space. This yields $\pi : \mathcal{I} \rightarrow U$. The components of a planning problem for information spaces on countable state spaces are:

- A nonempty *state space*, X, which is either finite or countably infinite.
- A finite *action space*, U. It is assumed that U contains the special termination action u_F.
- A finite *nature action space*, $\Theta(x, u)$ for each $x \in X$ and $u \in U$.
- A *state transition equation*, f, that produces a state, $f(x, u, \theta)$ for every $x \in X$, $u \in U$, and $\theta \in \Theta(x, u)$.
- A finite or countably infinite *observation space*, Y.
- A finite *nature observation action space*, $\Psi(x)$ for each $x \in X$.

- A *sensor mapping*, h.
- An *initial condition*, η_0.
- A *goal set*, $X_G \subseteq X$.
- A real-valued additive cost functional L, which may be applied to any state-action history, $(\tilde{x}_{K+1}, \tilde{u}_K)$, to yield

$$L(\tilde{x}_{K+1}, \tilde{u}_K) = \sum_{k=1}^{K} l(x_k, u_k) + l_F(x_{K+1}). \tag{41}$$

  If the termination action, u_F, is applied at some stage k, then for all $i \geq k$, $u_i = u_F$, $x_i = x_k$, and $l(x_i, u_F) = 0$ if $x_i \in X_G$, or ∞ otherwise.

As before, the cost functional $L(\tilde{x}, \tilde{u})$ allows the evaluation of the quality of a plan. Since there is uncertainty in the state prediction and in the sensing, we can use either worst-case or expected-case analysis for evaluating plans. If $\mathcal{H}(\pi, \eta_0)$ denotes the set of all possible state-action histories given the plan π from the initial condition, the cost of the plan with worst-case analysis is

$$G_\pi = \max_{(\tilde{x}, \tilde{u}) \in \mathcal{H}(\pi, \eta_0)} L(\tilde{x}_{K+1}, \tilde{u}_K). \tag{42}$$

If a probabilistic model of the uncertainty is known, the expected cost of a plan is

$$G_\pi = E_{\mathcal{H}(\pi, \eta_0)} L(\tilde{x}_{K+1}, \tilde{u}_K). \tag{43}$$

3.2 Deriving Information States

In its original definition, the information space seems unmanageable. In fact, it only seems useful for planning problems where the number of states is very small, since the history representing an information state grows linearly with the number of stages. The main idea here is to map the original information space into a smaller space, ensuring that when a successful plan exists over the original space, a plan will exist also in the smaller space. As expected, in the general case, the smaller space will present plans that are feasible, but may not be optimal in the original space. For most of the planning problems asking for a feasible plan is already a challenging task.

In general, let $\kappa : \mathcal{I} \rightarrow \mathcal{I}^\circ$ denote a surjective mapping from an information space $\mathcal{I}$ to a *derived information space*, $\mathcal{I}^\circ$. Ideally, $\mathcal{I}^\circ$ should be as small as possible while ensuring that solutions to the planning problem exist. While the design of the mapping κ may take advantage of specific planning problem characteristics, we next present two general approaches to derive information states for $\mathcal{I}^\circ$.

Nondeterministic Derived Information States. The first method we discuss is based on the inferences that can be done given an information state. If the information state η_k is available, it is possible to compute a the set $X_k(\eta_k)$ in which the actual x_k is known to lie. The set $X_k(\eta_k)$ is called a *derived information state*. To compute the derived information state, we have to infer over the observations and actions performed. For the observations, we can define

$$H(y) = \{x|\ y = h(x, \psi), \text{for } \psi \in \Psi(x)\} \tag{44}$$

that is, the set of all possible states the robot may be in given an observation. The set $H(y)$ is called the *preimage* of y. Similarly, if we let the actions available depend on the current state, the robot can determine a set of states V where it may be, by computing

$$V(U_k) = \{x'|U_k = U(x') \text{ for } x' \in X\}, \tag{45}$$

in which U_k are the actions available at stage k. The current state then lies in the set $H \cap V$. Note, however, that it can be assumed that the robot has some kind of sensor that detects which kind of actions are available. This reduces the computation of V and H into only the computation of H. Thus, we will discuss only the case when U will be fixed for all $x \in X$.

From the state transition equation, it is possible to know which states may be reached if action u is applied at state x. Let F be this set, formally defined as

$$F(x, u) = \{x' \in X|\exists\theta \in \Theta(x, u) \text{ for which } x' = f(x, u, \theta)\}. \tag{46}$$

Using F and H, we next present how to compute the derived information state, $X_k(\eta_k)$, for any state k, using induction. Note that F and H eliminate the direct appearance of nature actions. The base case

($k = 1$) of the induction is

$$X_1 = \eta_0 \cap H(y_1). \tag{47}$$

This first step consists only of making consistent the initial condition with the first observation. Now assume inductively that $X_k(\eta_k) \subseteq X$ is available, and $X_{k+1}(\eta_{k+1})$ should be computed. First note that $\eta_{k+1} = (\eta_k, u_k, y_{k+1})$, and the new information is provided only by u_k and y_{k+1}. From (44), the state is anywhere in $H(y_{k+1})$. On the other hand, if x_k was known, after applying u_k, the state lies somewhere in $F(x_k, u_k)$. Since x_k is unknown, but it is known that $x_k \in X_k(\eta_k)$, the new derived information state is

$$X_{k+1}(\eta_k, u_k, y_{k+1}) = \bigcup_{x_k \in X_k(\eta_k)} F(x_k, u_k) \cap H(y_{k+1}). \tag{48}$$

Given that the derived information state is always a subset of X, the derived information space can be defined as $\mathcal{I}^\circ = 2^X$. Note that if X is finite, $\mathcal{I}^\circ$ is also finite, which makes it preferable if the number of stages is much larger than the size of X.

Probabilistic Derived Information States. As before, we will compute derived information states, but assuming that nature is modeled probabilistically. Nature is also assumed to follow a Markov model, in which its actions depend only on the current state, as opposed to actions or state histories. Thus, a derived information state becomes a conditional probability distribution. The set functions H and F become $P(x_k|y_k)$ and $P(x_{k+1}|x_k, u_k)$, respectively. To compute $P(x_k|y_k)$ Bayes rule is applied as:

$$P(x_k \cap y_k) = P(x_k|\ y_k)P(y_k) = P(y_k|\ x_k)P(x_k). \tag{49}$$

Solving for $P(x_k|y_k)$ yields

$$P(x_k|y_k) = \frac{P(y_k|x_k)P(x_k)}{P(y_k)} = \frac{P(y_k|x_k)P(x_k)}{\sum_{x_k \in X} P(y_k|x_k)P(x_k)}. \tag{50}$$

Bayes' rule requires the knowledge of $P(x_k)$ and $P(y_k|\ x_k)$. The prior $P(x_k)$ will be replaced later by a derived information state, while the probability $P(y_k|x_k)$ is easily computed as

$$P(y_k|x_k) = \sum_{\psi \in \Psi(x_k): y_k = h(x_k, \psi)} P(\psi|x_k). \tag{51}$$

Since each information state is a probability distribution over X, it can be written as $P(x_k|\ \eta_k)$, if it is derived from η_k. As before, derived information states can be computed inductively. For the base case $(k = 1)$ we have $\eta_0 = P(x_1)$ and the first observation y_1. Together they determine $P(x_1|y_1)$ as

$$P(x_1|\ \eta_1) = P(x_1|\ y_1) = \frac{P(y_1|\ x_1)P(x_1)}{\sum_{x_1 \in X} P(y_1|x_1)P(x_1)}. \tag{52}$$

Assuming inductively that $P(x_k|\eta_k)$ has been computed, $P(x_{k+1}|\eta_{k+1})$ has to be determined. Once again the derived information state can be written as $P(x_{k+1}|\eta_k, u_k, y_{k+1})$. Considering first the effect of u_k, note that

$$P(x_{k+1}|\eta_k, x_k, u_k) = P(x_{k+1}|x_k, u_k), \tag{53}$$

because η_k contains no additional information regarding the prediction of x_{k+1} when x_k is given. To eliminate x_k from $P(x_{k+1}|x_k, u_k)$ marginalization is used, giving the derived information state

$$P(x_{k+1}|\eta_k, u_k) = \sum_{x_k \in X} P(x_{k+1}|x_k, u_k, \eta_k)P(x_k|\eta_k) = \sum_{x_k \in X} P(x_{k+1}|x_k, u_k)P(x_k|\eta_k). \tag{54}$$

The next step is to take into account the observation, y_{k+1}. From (50), k is replaced with $k + 1$ and $P(x_k)$ is replaced with the information accumulated, to give

$$P(x_{k+1}|\ y_{k+1}, \eta_k, u_k) = \frac{P(y_{k+1}|x_{k+1}, \eta_k, u_k)P(x_{k+1}|\eta_k, u_k)}{\sum_{x_{k+1} \in X} P(y_{k+1}|x_{k+1}, \eta_k, u_k)P(x_{k+1}|\eta_k, u_k)}. \tag{55}$$

The expression for $P(x_{k+1}|\eta_k, u_k)$ was given in (54). To calculate $P(y_{k+1}|x_{k+1}, \eta_k, u_k)$ note that

$$P(y_{k+1}|x_{k+1}, \eta_k, u_k) = P(y_{k+1}|x_{k+1}) \tag{56}$$

because the observation depends only on the state[2]. Since $P(y_{k+1}|x_{k+1})$ is given as part of the sensor model, we are finished deriving the computation of $P(x_{k+1}|\eta_{k+1})$ from $P(x_k|\eta_k)$.

In this case, the derived information space is the set of all probability distributions over X. Thus, the planning problem can be expressed again entirely in terms of the derived information space. A goal region can be specified as constraints on the probabilities. For example, for some particular $x \in X$, the goal might be to reach any derived information state for which $P(x|\eta_k) > 0.9$.

Let $n = |X|$. It is possible to embed $\mathcal{I}^\circ$ in $\mathbb{R}^n$ with each state $x \in X$ representing a vertex of a $(n-1)$-simplex. The coordinates of each vertex are expressed using probabilities $(p_1, p_1, \ldots, p_n)$ as barycentric coordinates. Here p_i is the probability of being in state x_i. Since $p_1 + \cdots + p_n = 1$, the vertices of the simplex (i.e., $(1, 0, \ldots, 0)$, $(0, 1, \ldots, 0)$, $\cdots$, $(0, 0, \ldots, 1)$) correspond to the cases when the state is completely known. A planning problem of this kind is known as a *Partial Observable Decision Process* (POMDP).

Efficient solutions to POMDPs form an active area in the research community [77, 78]. The problem is clearly very difficult, since the dimension of the space grows linearly with the number of states. However, the method of value iteration, presented in Section 2.3.1 can be applied. Let $\vec{x} \in \mathcal{I}_t$ be a derived information state. A worst-case analysis yields a cost functional of

$$\vec{l}(\vec{x}_k, u_k) = \max_{x_k \in X_k(\eta_k)} l(x_k, u_k) \tag{57}$$

and

$$\vec{l}_F(\vec{x}_F) = \max_{x_F \in X_F(\eta_F)} l_F(x_F). \tag{58}$$

Thus, the dynamic programming recursion is similar to the one presented in Section 2.3.1, but using derived information states:

$$G_k^*(\vec{x}_k) = \min_{u_k \in U} \left\{ \vec{l}(\vec{x}_k, u_k) + \sum_{\vec{x}_{k+1} \in \mathcal{I}_t} G_{k+1}^*(\vec{x}_{k+1}) P(\vec{x}_{k+1}|\vec{x}_k, u_k) \right\}. \tag{59}$$

Note that the set of observations and nature actions is finite, since $\mathcal{I}^\circ$ is finite. This implies that $P(\vec{x}_{k+1}|\vec{x}_k, \vec{u}_k)$ is only an approximation distributed over a finite set of points of $\mathcal{I}_t$. The space $\mathcal{I}^\circ$ is a continuous space which usually requires the specification of a probability density function.

[2]Here we are assuming that the sensor mapping does not depend on the history

A policy can be found by approximating with a grid in the $(n-1)$-simplex, and using interpolation for evaluating points not in the grid [79,80]. This method will be described in more detail for information spaces when the state space is continuous.

3.3 Continuous State Spaces

Until now, we have described information spaces when the underlying state space X is countable. Now we consider the case when X is a continuous space.

3.3.1 Sensors

As expected, the catalog of sensors is richer in the continuous case. Some models of sensors are:

- **Linear sensing.** It is assumed that $Y = X$. Thus, an *identity sensor* can be defined in which $y = h(x)$ makes the state immediately known. If there is a bound r in the error of the measurement, the state lies in the ball of radius r centered at y. This error is also commonly modeled with a probability distribution (i.e., a Gaussian).

- **Projection.** In this model the dimension of the observation space, n_y, is smaller than the dimension of the state space. Either the observations ignore coordinates of X (i.e., a gyroscope gives orientation, but ignores position), or X is embedded in a smaller dimensional space (i.e., a photograph takes $X \subset \mathbb{R}^3$ into $\mathbb{R}^2$).

- **Landmark sensor.** A landmark sensor detects specific identifiable features in the environment. In its more abstract form, it detects specific points in the space (i.e., goal points or regions).

Specific sensors, such as an *odometry sensor*, which gives an estimation of the distance traveled, can be defined in terms of a projection sensor modeled together with a history-based sensor mapping. In recent years, *depth sensors* have been widely used in mobile robotics. This type of sensors gives measurements of the spatial distribution and shape of the obstacles in the environment. This accounts for sensors such as the *sonar*, or the *laser range finder*. Each sensor has an *upper range*. Obstacles farther from the sensor than this range cannot be detected. As the range is decreased, the sensor becomes a *proximity* sensor, and in the limit case it becomes a *contact* sensor. Note that the physical implementation may vary widely here. While

an acoustic sonar measure time of flight of a high frequency sound, the contact sensor may be a device that makes a reading when it is *pushed*, thus indicating that distance is equal to 0.

3.3.2 Discrete-Stage Information Spaces

The simpler case corresponds to a plan with discrete stages, and many of the concepts for discrete spaces, at least at first glance, are the same as their continuous counterparts. Let the state space $X \subset \mathbb{R}^m$ be an n-dimensional manifold[3]. The *observation space* $Y \subseteq \mathbb{R}^m$ is now an n_y-dimensional manifold, for $n_y \leq m$. Also, let $U \subseteq \mathbb{R}^m$ be an n_u-dimensional manifold for $n_u < m$.

Given that the time is discrete, the concepts presented for discrete spaces in Section 3.1.2 remain the same, but taking into account the fact that the variables are continuous.

3.3.3 Continuous-Time Information Spaces

Most of the definitions presented in Section 3.3.2 remain the same when we consider a continuum of stages. Thus, X, Y, $\Psi(x)$ and $\Theta(x,u)$ are defined as before. However, the state transition equation now takes the form

$$\frac{\partial x}{\partial t} = \dot{x} = f(x, u, \theta), \tag{60}$$

for $x \in X$, $u \in U$ and $\theta \in \Theta(x,u)$. This means that the nature actions $\Theta(x,u)$ should be expressed in terms of velocities. Also, in the discrete case, an information state was expressed in terms of history sequences, but in the continuous case, histories become a function of time. Thus, $\tilde{y}_t : [0,t) \rightarrow Y$ $\tilde{u}_F : [0,t) \rightarrow U$, and $\tilde{x}_F : [0,t) \rightarrow X$ are the *observation history*, *action history*, and *state history*, respectively, up to time t.

The sensor mappings are now expressed with:

1. **State-sensor mapping.** $y(t) = h(x(t))$.
2. **State-nature mapping.** $y(t) = h(x(t), \psi(t))$
3. **History-based sensor mapping.** $y(t) = h(\tilde{x}_F, \psi(t))$.

Note that $\tilde{x}$ is usually the solution of a differential equation.

[3]For readers unfamiliar with the term, an n-manifold is a space that locally looks like $\mathbb{R}^n$. Our everyday notion of a surface corresponds to a 2-manifold as a subset of $\mathbb{R}^3$. See [81].

The *information state at time t* becomes

$$\eta_t = (\eta_0, \tilde{u}_t, \tilde{y}_t), \tag{61}$$

which has the same form and meaning as its discrete counterpart, but in continuous time. The set of all possible η_t is the *information space at time t*, $\mathcal{I}_t$. Since each $\eta_t \in \mathcal{I}_t$ is a function of time, $\mathcal{I}_t$ is a space of functions. Combining all the information spaces up to time $T \in [0, \infty)$, a single information space $\mathcal{I}$ is obtained as

$$\mathcal{I} = \bigcup_{t \in T} \mathcal{I}_t. \tag{62}$$

To evaluate the quality of a plan, a new cost functional should be defined. Let L denote a real-valued, additive cost functional, which may be applied to any state-action history, $(\tilde{x}_t, \tilde{u}_t)$, defined as

$$L(\tilde{x}_t, \tilde{u}_t) = \int_0^{t'} l(x(t'), u(t'))dt' + l_F(x(t)), \tag{63}$$

in which $l(x(t'), u(t'))$ is the instantaneous cost, and $l_F(x(t))$ is a final cost.

3.4 Examples of Planning in the Information Space

3.4.1 Moving in an L-shaped corridor

This idealized example, which appeared originally in [23], is intended to illustrate the issues that arise in selecting an appropriate map κ for derived information states. The state space, X, for the example shown in Figure 11 has 19 states, each of which corresponds to a location on one of the white tiles. For convenience, let each state be denoted by (i, j). There are 10 *bottom states*, denoted by $(1, 1)$, $(2, 1)$, ..., $(10, 1)$, and 10 *left states*, denoted by $(1, 1)$, $(1, 2)$, ..., $(1, 10)$. Since $(1, 1)$ is both a bottom state and a left state, it will be called the *corner state*.

It is assumed for this problem that there are no sensor observations. Nature, however, interferes with the state transitions, which leads to a form of nondeterministic uncertainty. If we try to apply an action that takes one step, nature may cause two or three steps to be taken, if possible. This can be modeled as follows. Let $U = \{(1, 0), (-1, 0), (0, 1), (0, -1)\}$ and let $\Theta = \{1, 2, 3\}$. The state transition equation is defined as $f(x, u, \theta) = x + \theta u$, unless it is impossible to move to the required location, in which case $f(x, u, \theta) = x$. For

example, if $x = (5, 1)$, $u = (-1, 0)$, and $\theta = 2$, then the resulting next state is $(5, 1) + 2(-1, 0) = (3, 1)$.

Since there are no sensor observations, the information state at stage k is

$$\eta_k = (u_1, \ldots, u_{k-1}). \tag{64}$$

Now use the derived information space, $\mathcal{I}^\circ = 2^X$. The initial state, $x_1 = (10, 1)$ is given, which means that the initial information state, η_1, is $\{(10, 1)\}$. The goal is to arrive at the information state, $\{(1, 10)\}$, which means that the task is to design a plan that moves from the lower right to the upper left.

With perfect information, this would be trivial; however, without sensors the uncertainty may grow very quickly. For example, after applying the action $u_1 = (-1, 0)$ from the initial state, the derived information state becomes $\{(7, 1), (8, 1), (9, 1)\}$. After $u_2 = (-1, 0)$ it becomes $\{(4, 1), \ldots, (8, 1)\}$. A nice feature of this problem, however, is that uncertainty can be reduced without sensing. Suppose that for 100 stages, we continue to apply $u_k = (-1, 0)$. What is the resulting information state? As the corner state is approached, the uncertainty is reduced because the state cannot be further changed by nature. It is known that each action, $u_k = (-1, 0)$, decreases the X coordinate by at least one each time. Therefore, after 9 or more stages, it is known that $\eta_k = \{(1, 1)\}$. Once this is known, then the action $(0, 1)$ can be applied. This will again increase uncertainty as the state moves through the set of left states. If $(0, 1)$ is applied 9 or more times, then it is known for certain that $x_k = (1, 10)$, which is the required goal state.

A successful plan has now been obtained: apply $(-1, 0)$ for 9 stages, then apply $(0, 1)$ for 9 stages. Recall from Section 3.1.2 that a strategy is generally specified as $\pi : \mathcal{I} \rightarrow U$; however, for this example, it appears that only a sequence of actions is needed. The actions do not depend on the information state. Why did this happen? If no observations are obtained during execution, then there is no way to use feedback. There is nothing to learn by executing the plan. In general, for problems that involve no sensors and a fixed initial information state, a *path* in the information space can be derived from a plan. It is somewhat strange that this path is completely predictable, even though the original problem may involve substantial uncertainties. We always know precisely what will happen in terms of the information states if there are no sensors and the initial condition is fixed.

To make the situation more interesting, assume that any subset of X could be used as the initial condition. In this case, a plan $\pi : \mathcal{I} \rightarrow U$ must be formulated to solve the problem. From each initial information state

η, a path in $\mathcal{I}$ can still be computed from π. Specifying a plan over all of $\mathcal{I}$ appears complicated, which motivates the next consideration.

The ideas from Section 3.2 can be applied here to collapse the information down from 2^{19} (over half of a billion) to 19 derived information states. The mapping $\kappa : \mathcal{I} \rightarrow \mathcal{I}^\circ$ must be constructed. We first make a naive attempt to collapse the information state down to only three states. Let $\mathcal{I}^\circ = \{g, l, a\}$, in which g denotes "goal", l denotes "left", and a denotes "any". The mapping is

$$\kappa(\eta) = \begin{cases} g & \text{if } \eta = \{(1, 10)\} \\ l & \text{if } \eta \text{ is a subset of the set of left states} \\ a & \text{otherwise} \end{cases} . \tag{65}$$

It might seem that this derived information space will lead to a very compact plan for solving the problem. Based on the successful plan described so far, the plan on $\mathcal{I}^\circ$ can be defined as $\pi(g) = u_F$, $\pi(l) = (0, 1)$, and $\pi(a) = (-1, 0)$. What is wrong with this? Suppose that the initial state is $(10, 1)$. There is no way to require that $u_k = (-1, 0)$ be applied 9 times to reach the l state. If $(-1, 0)$ is applied to the a state, then it is not possible to determine when the transition to l should occur.

Now consider a different derived information space. Suppose that there are 19 derived information states, which includes g as defined previously, l_i for $1 \leq j \leq 9$, and a_i for $2 \leq i \leq 10$. The mapping κ is defined as $\kappa(\eta) = g$ if $\eta = \{(1, 10)\}$. Otherwise, $\kappa(\eta) = l_i$, for the smallest value of i such that η is a subset of $\{(1, i), \ldots, (1, 10)\}$. If there is no such value for i, then $\kappa(\eta) = a_i$, for the smallest value of i such that η is a subset of $\{(1, 1), \ldots, (1, 10), (2, 1), \ldots, (i, 1)\}$. Now the plan may be defined as $\pi(g) = u_F$, $\pi(l_i) = (0, 1)$, and $\pi(a_i) = (-1, 0)$. Although it might not appear to be any better than the plan obtained from collapsing $\mathcal{I}^\circ$ to three states, the important difference is that the correct information state transitions occur. For example, if $u_k = (-1, 0)$ is applied at a_5, then a_4 is obtained. If $u = (-1, 0)$ is applied at a_2, then l_1 is obtained. From there, $u = (0, 1)$ is applied to yield l_2. These actions can be repeated until eventually l_9 and g are reached.

3.4.2 The Kalman filter

When the transition function f, and the sensor mapping h are both linear functions, and nature actions, θ and ψ, can be modeled as Gaussian, the derived information states will follow a Gaussian distribution too. These assumptions are reasonable in many mobile robotics contexts. In this case, a mapping $\kappa : \mathcal{I} \rightarrow \mathcal{I}^\circ$, in

which $\mathcal{I}^\circ$ is the space of all Gaussians, will collapse $\mathcal{I}$ without any loss of information. This is referred to as a *linear-Gaussian* model, which is the basis for the most common approach for collapsing $\mathcal{I}$, the *Kalman filter*. Each Gaussian is specified by an n-dimensional mean vector μ, and an $n \times n$ symmetric covariance matrix, Σ.

Since the Kalman filter relies on linear models, f can be written as

$$x_{k+1} = A_k x_k + B_k u_k + G_k \theta_k, \tag{66}$$

in which A_k, B_k, and G_k are real-valued matrices of appropriate dimensions. The subscript k is used because the Kalman filter works even if f is different in every stage. Similarly, the sensor mapping becomes

$$y_k = C_k x_k + H_k \psi_k. \tag{67}$$

Since an information state $P(x_k|\eta_k) \in \mathcal{I}^\circ$ is represented by its mean vector and its covariance matrix, our goal here is to compute μ_k and Σ_k at stage k. We next give the update expressions, omitting their derivation, that can be found in many textbooks on stochastic control (i.e., [82]). Given the initial conditions μ_0 and Σ_0, we have

$$\Sigma'_{k+1} = A_k \Sigma_k A_k^T + G_k \Sigma_\theta G_k^T, \tag{68}$$

$$\Sigma_{k+1} = (I - L_{k+1} C_{k+1}) \Sigma'_{k+1}, \tag{69}$$

$$\mu_{k+1} = A_k \mu_k + L_{k+1}(y_{k+1} - C_{k+1} A_k \mu_k), \tag{70}$$

with

$$\Sigma_{k+1} = (I - L_{k+1} C_{k+1}) \Sigma'_{k+1}. \tag{71}$$

The expression for L_k (substitute $k+1$ for k to obtain L_{k+1}) is

$$L_k = \Sigma'_k C_k^T [C_k \Sigma'_k C_k^T + H_k \Sigma_\psi H_k]^{-1}. \tag{72}$$

When nature is not Gaussian, or the transition equation is not linear, the derived information states density can be approximated using a grid, with numerical integration between the grid points. Let $S \subset X$

be the set of points in the grid. In the initial step, $P(s)$ is computed from $p(x)$ by numerically evaluating the integrals of $p(x_1)$ over the Voronoi region of each sample. Now suppose that $P(s_k|\eta_k)$ has been computed over S_k, and the task is to compute $P(s_{k+1}|\eta_{k+1})$ given u_k and y_{k+1}.

Considering only u_k, $P(s_{k+1}|s_k, u_k))$ approximates $p(x_{k+1}|x_k, u_k)$ when computed in the manner described above. At this point the densities needed have been approximated by discrete distributions.

The resulting distribution is $P(s_{k+1}|\eta_k, u_k)$, and the effect of y_{k+1} in $p(x_{k+1}|y_{k+1})$ can be computed approximately by $P(s_{k+1}|y_{k+1})$ using the grid samples. The resulting distribution, $P(s_{k+1}|\eta_{k+1})$ represents the approximate derived information state. It turns out that the Voronoi regions over the samples do not even need to be carefully considered. One can work directly with a collection of samples randomly drawn from the initial probability density $p(x_1)$. The general method is referred to as *particle filtering*, and has yielded good performance in applications to experimental mobile robotics [83].

3.4.3 Sensorless manipulation

Imagine a planning problem in which the robot does not have any sensors, so that there are no observations at all. Moreover, the initial condition is unknown. Is it still possible to compute a plan to reach a goal state? As we will explore in the next example, in some problems knowing only the action history is enough to compute a successful plan.

In the context of manufacturing, a *part* may need to have an specific orientation before being assembled with other components. In a sensorless setting, a robot, in this case a robotic arm with a gripper, needs to orient a part without any feedback [84, 85]. The part is modeled as a convex polygon. Its initial orientation is unknown; the goal is to bring the part to a known orientation, up to symmetry. The manipulation process is shown in Fig. 8. The part moves on the conveyor towards a fence, against which it comes to rest after possibly rotating to reach a stable orientation. The robotic arm grasps the part, changes its orientation, and drops it up again in the conveyor. This process is repeated until the part achieves the desired orientation against the fence.

The natural state space for this problem is S^1, corresponding to the orientation of the part. At each step, the robotic arm rotates the part through some angle, so the action space is likewise S^1. These continuous spaces need to be transformed into finite sets. The key of the transformation is to identify *critical events* which partition the space into equivalence classes, then plan over this set of equivalence classes rather than

the full space. These critical events are problem specific. In the case of the part orienting, the critical events in action space are orientation angles such that for a given information state, rotations either greater or less than these angles will reach different information states.

Thus, the state space X is chosen as the set of all the stable orientations of the part when it is lying statically on the fence. Since the part is polygonal, the size of X is bounded above by the number of edges in the part. Using the concepts presented in Section 3.2, the derived information space is $\mathcal{I}^\circ = 2^X$. The initial derived information state consists of all possible stable orientations (i.e. $\eta_0 = X$), since the part orientation is initially unknown. The action set is the range of rotation angles available to the gripper, partitioned into intervals of rotations that lead to identical resulting information states. The effects of a specific rotation action on a rectangular part are shown in Fig. 9. The critical events in the continuous action set for an information state with two states are shown in Fig. 10.

The objective is to find a sequence of actions such that the derived information state at the final stage corresponds to a single possible orientation of the part. Once one orientation is uniquely identified, the robotic arm may perform an additional rotation to achieve any desired goal orientation. With the finite action set, a directed graph can be constructed whose nodes are information states and whose edges are transitions resulting from the discrete action set. Standard graph searching techniques can be used to search for a directed path to a singleton information state. This path in the collapsed information space graph constitutes a plan for eliminating uncertainty in the part's orientation.

The reason that successful planning is still possible starting from total uncertainty and without sensor feedback is that some actions in this information space have a *conformant* property, in which the same resulting state can be reached by the same action from many different initial states. By selecting conformant actions, uncertainty can be reduced. The same principle is applied in the context of mobile robot localization with extremely limited sensing in [75].

4 Conclusion and Bibliographical Remarks

Our presentation has been tightly constrained by space limitations. There are a number of books to which we refer the reader for elaboration. Treatments of decision theory in general appear in [4–6]. Bertsekas [24] covers much of the same material as the present chapter and is well-stocked with examples. Part III of [23]

is also quite similar in content but is much more detailed. A general treatment of the infinite horizon case is [43]. Sutton and Barto [46] is the definitive introduction to reinforcement learning. Ghallib, Nau and Traverso [86] consider planning with primarily logic-based representations. Russell and Norvig [87] cover planning under (mainly probabilistic) uncertainty from an artificial intelligence perspective. Some recent papers on decision-theoretic planning are collected in [88]. Game theory is addressed in greater detail in [47, 48, 89].

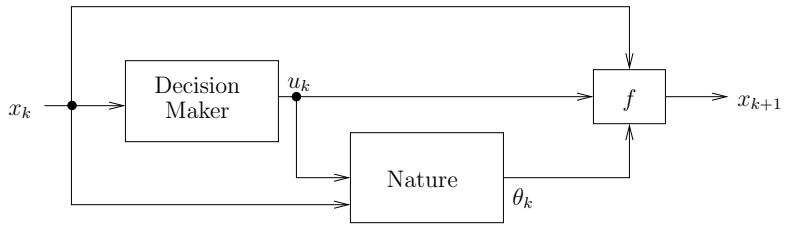

Figure 1: Planning with prediction uncertainty.

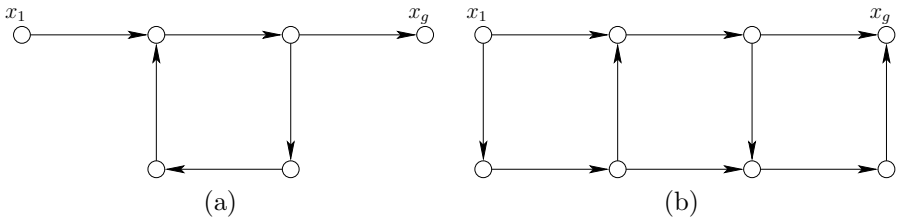

Figure 2: Two simple nondeterministic planning problems. In (a), nature can always prevent the decision maker from reaching the goal. In (b), all flows lead to the goal.

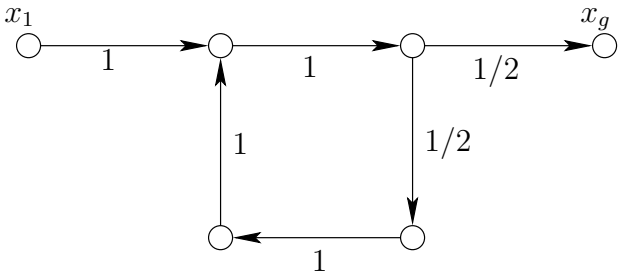

Figure 3: Probabilistic uncertainty can cause value iteration to converge only in the limit. Edges are labeled with transition probabilities. In this example, nature can cause executions of arbitrary length. However, executions that traverse the cycle in the graph many times are unlikely. Assuming the cost of each transition is 1, the cost-to-go for x_1 converges $2 + 4 \sum_{i=0}^{\infty} i \left(\frac{1}{2}\right)^{i+1} + 1 = 7$.

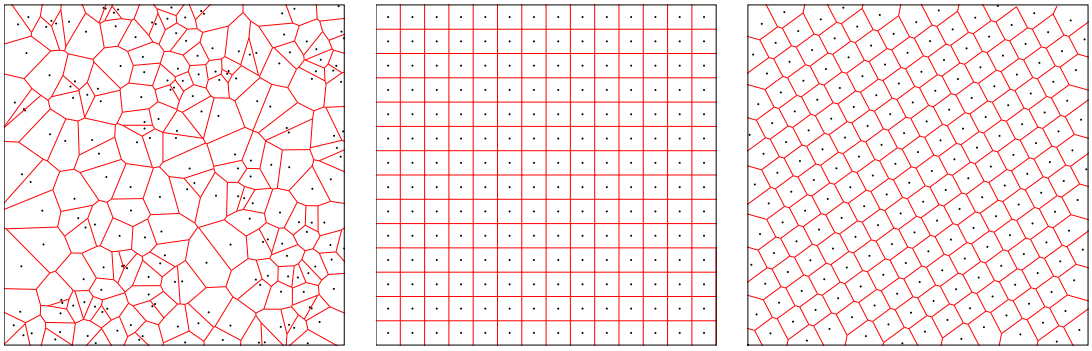

Figure 4: Three ways to select samples in the unit square. Dots represent samples, the lines show their respective Voronoi cells. Larger Voronoi cells indicate poor uniformity. (a) Pseudo-random samples. (b) Grid samples. (c) Lattice samples.

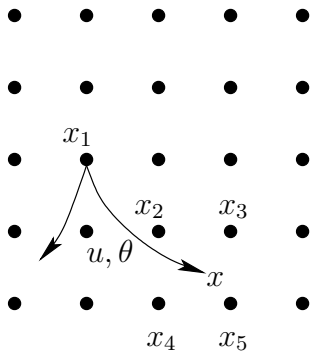

Figure 5: Dots denote sample states. Applying an action u and nature action θ on a sampled state x_1 moves the system to state x which is not in the sampled set. Some form of interpolation is needed to estimate the value function at x.

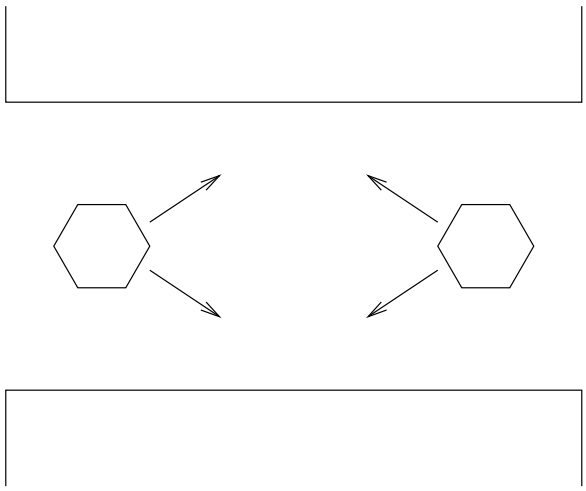

Figure 6: An illustration of the need for mixed strategies. The left robot attempts to pass through a corridor while the right robot attempts to block its progress. Each must independently decide whether to move to the left or right. If the left robot plays a pure strategy, the right robot can take advantage and always get in the way. A mixed strategy that chooses each direction equally often will enable the left robot to escape.

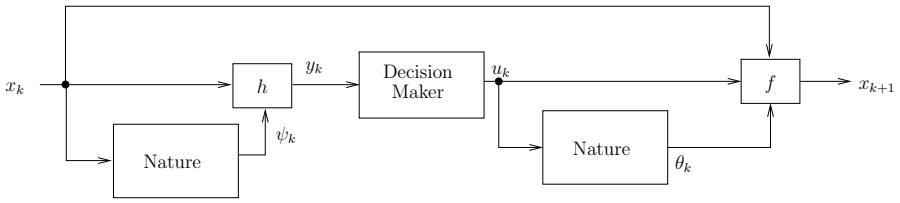

Figure 7: Planning with sensing uncertainty. Note that the decision maker does not have direct access to the state.

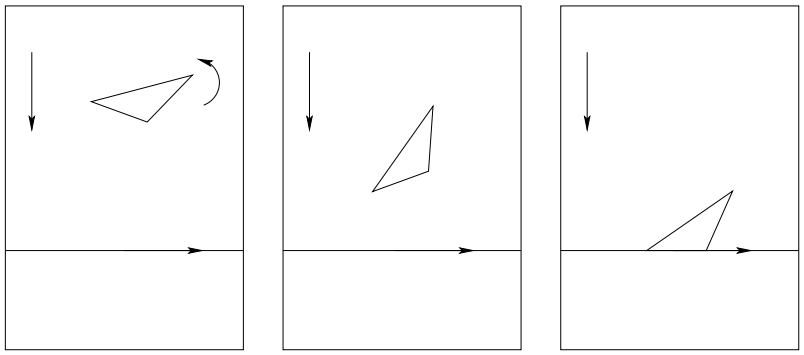

Figure 8: Overhead overview of a part on the conveyor. The conveyor moves downward. The robot picks up a part and rotates it through a chosen angle before placing it on the conveyor. The part then drifts on the conveyor into contact with the fence, possibly rotating compliantly as it comes to rest.

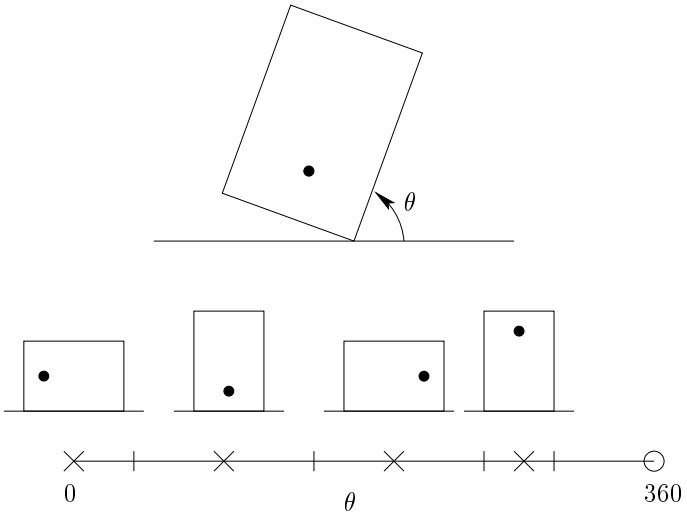

Figure 9: Effects of rotation actions on a rectangular part. The action space is divided into four equivalence classes according to the resulting state. The crosses mark a representative action from each class.

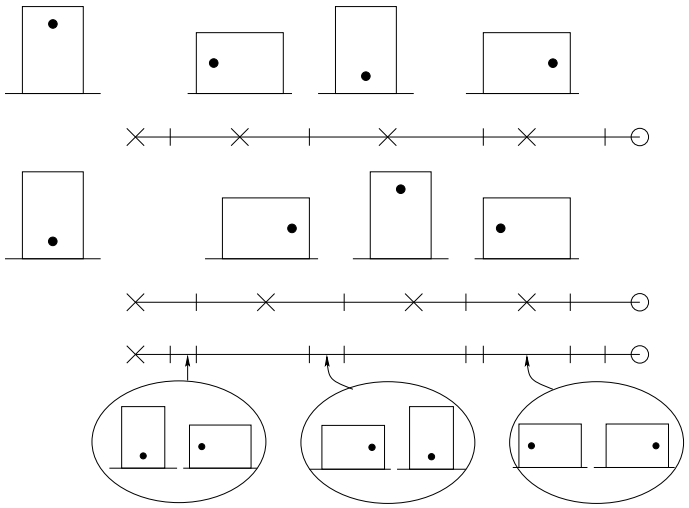

Figure 10: Critical events for an information state with two states.

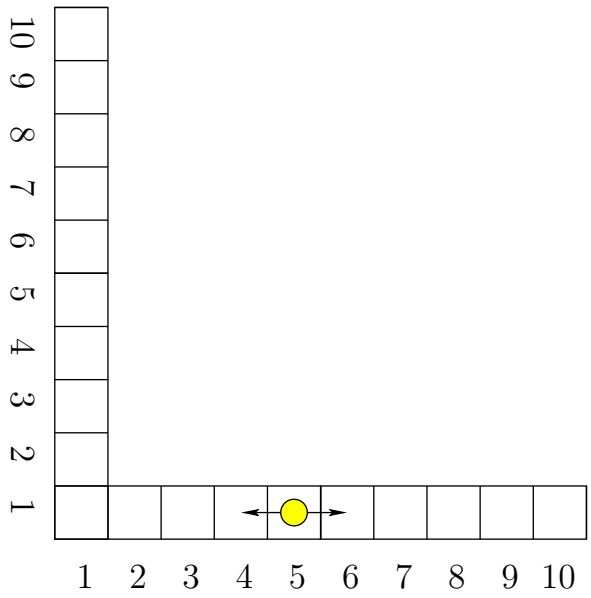

Figure 11: An example that involves 19 states. There are no sensor observations; however, actions can be chosen that that enable the state to be estimated. The example provides an illustration of collapsing the information space.

Authors' Biographies

Jason M. O'Kane is a Ph.D. candidate in the Department of Computer Science at the University of Illinois at Urbana-Champaign. In 2001, he received a Bachelor of Science degree from Taylor University in Upland, Indiana. In 2005, he received a Master of Science degree from the University of Illinois at Urbana-Champaign. His research is in geometric algorithms for robotics.

Benjamín Tovar received the B.S. degree in electrical engineering from ITESM at Mexico City, Mexico, in 2000, and the M.S. in electrical engineering from University of Illinois, Urbana-Champaign, USA, in 2004. Currently (2005) he is pursuing the Ph.D. degree in Computer Science at the University of Illinois. Prior to M.S. studies he worked as a research assistant at Mobile Robotics Laboratory at ITESM Mexico City. He is mainly interested in motion planning, visibility-based tasks, and minimal sensing for robotics.

Peng Cheng is a Ph.D. candidate in Department of Computer Science at University of Illinois at Urbana-Champaign. He received his Bachelor and Master of Engineering degrees in Electrical Engineering from Tsinghua University, Beijing, China in 1996 and 1999, respectively. In 2001, he graduated with a Master of Science degree in Computer Science from Iowa State University. He has worked in the areas of motion planning and robotics, especially on problems with real or virtual robotic systems under differential constraints, which include how to represent, characterize, plan, and control the motion of these systems.

Steven M. LaValle is an Associate Professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He received his Ph.D. in Electrical Engineering in 1995 from the University of Illinois. From 1995-1997, he was a post-doctoral researcher and lecturer in the Department of Computer Science at Stanford. From 1997-2001, he was an Assistant Professor at Iowa State University. In 1999 he received the CAREER award from the US National Science Foundation. He has published in the areas of robotics, computational geometry, artificial intelligence, computational biology, computer vision, and control theory. He recently authored an on-line book, *Planning Algorithms*, which will be published by Cambridge University Press.

References

[1] K. Arrow and L. Hurwicz. An optimality criterion for decision-making under ignorance. In C.F. Carter and J.L. Ford, editors, *Uncertainty and expectation in economics*, Oxford, UK, 1972. Basil Blackwell and Mott Ltd.

[2] A. Wald. *Statistical Decision Functions*. J. Wiley and Sons, New York, 1950.

[3] J. Von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ, 1944.

[4] J. O. Berger. *Statistical Decision Theory*. Springer-Verlag, Berlin, 1980.

[5] M. H. DeGroot. *Optimal Statistical Decisions*. McGraw Hill, New York, NY, 1970.

[6] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice-Hall Publications, Englewood Cliffs, NJ, 1982.

[7] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, second edition, 2000.

[8] T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[9] S. Barbera, P.J. Hammond, and C. Seidl, editors. *Handbook of Utility Theory, volume 1: Principles*. Kluwer, Dordrecht, 1998.

[10] C. P. Robert. *The Bayesian Choice, 2nd Ed.* Springer-Verlag, Berlin, 2001.

[11] I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin. Claraty and challenges of developing interoperable robotic software. In *invited to International Conference on Intelligent Robots and Systems (IROS), Nevada*, October 2003.

[12] I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and Won Soo Kim. CLARAty: An architecture for reusable robotic software. In *SPIE Aerosense Conference, Orlando, Florida*, April 2003.

[13] C. Urmson, R. Simmons, and I. Nesnas. A generic framework for robotic navigation. In *Proceedings of the IEEE Aerospace Conference, Montana*, March 2003.

[14] K. Z. Haigh and M. M. Veloso. High-level planning and low-level execution: Towards a complete robotic agent. In W. Lewis Johnson, editor, *Proceedings of First International Conference on Autonomous Agents*, pages 363–370, Marina del Rey, CA, February 1997. ACM Press, New York, NY.

[15] R. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan. A layered architecture for office delivery robots. In W. Lewis Johnson, editor, *Proceedings of First International Conference on Autonomous Agents*, pages 245–252, Marina del Rey, CA, February 1997. ACM Press, New York, NY.

[16] Reid Simmons, Dani Goldberg, Adam Goode, Michael Montemerlo, Nicholas Roy, Brennan Sellner, Chris Urmson, Alan Schultz, Myriam Abramson, William Adams, Amin Atrash, Magda Bugajska, Michael Coblenz, Matt MacMahon, Dennis Perzanowski, Ian Horswill, Robert Zubek, David Kortenkamp, Bryn Wolfe, Tod Milam, and Bruce Maxwell. GRACE: an autonomous robot for the AAAI robot challenge. *AI Magazine*, pages 51–72, Summer 2003.

[17] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[18] R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA., 1960.

[19] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd ed)*. Johns Hopkins University Press, Baltimore, MD, 1996.

[20] D. P. Bertsekas. Distributed dynamic programming. *IEEE Transactions on Automatic Control*, 27:610–616, 1982.

[21] D. P. Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27:107–120, 1983.

[22] A. G. Barto, S. J. Bradtke, and S. P. Singh. Real–time learning and control using asynchronous dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[23] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. To appear in print. Available online at http://msl.cs.uiuc.edu/planning/.

[24] D. P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientic, Belmont, MA, second edition, 2001.

[25] D. H. Lehmer. Mathematical methods in large-scale computing units. In *Proc. 2nd Sympos. on Large-Scale Digital Computing Machinery*, pages 141–146. Harvard University Press, 1951.

[26] H. Niederreiter. *Random Number Generation and Quasi-Monte-Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1992.

[27] A. G. Sukharev. Optimal strategies of the search for an extremum. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 11(4), 1971. Translated from Russian, *Zh. Vychisl. Mat. i Mat. Fiz.*, 11, 4, 910-924, 1971.

[28] J. Matousek. *Geometric Discrepancy*. Springer-Verlag, Berlin, 1999.

[29] I. H. Sloan and S. Joe. *Lattice Methods for Multiple Integration*. Oxford Science Publications, Englewood Cliffs, NJ, 1990.

[30] X. Wang and F. J. Hickernell. An historical overview of lattice point sets. In K.-T. Fang, F. J. Hickernell, and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 158–167. Springer-Verlag, Berlin, 2002.

[31] S. Davies. Multidimensional triangulation and interpolation for reinforcement learning. In *Advances in neural information processing systems*, 1996.

[32] D. P. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[33] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[34] S. M. LaValle and P. Konkimalla. Algorithms for computing numerical optimal feedback motion strategies. *International Journal of Robotics Research*, 20(9):729–752, September 2001.

[35] P. Konkimalla and S. M. LaValle. Efficient computation of optimal navigation functions for nonholonomic planning. In *Proc. First IEEE Int'l Workshop on Robot Motion and Control*, pages 187–192, 1999.

[36] R. Munos and A. Moore. Barycentric interpolators for continuous space and time reinforcement learning. In *Neural Information Processing Systems*, volume 11. MIT Press, December 1998.

[37] D. Moore. *Simplical mesh generation with applications*. PhD thesis, Cornell University, 1992.

[38] R. E. Bellman and S. E. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, Princeton, NJ, 1962.

[39] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[40] R. E. Larson. A survey of dynamic programming computational procedures. *IEEE Trans. Autom. Control*, 12(6):767–774, December 1967.

[41] R. E. Larson and J. L. Casti. *Principles of Dynamic Programming, Part II*. Dekker, New York, NY, 1982.

[42] L.P. Kaelbling, M.L. Littman, and Andrew Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[43] D. P. Bertsekas. *Dynamic programming and optimal control*, volume 2. Athena Scientic, Belmont, MA, second edition, 2001.

[44] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[45] R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, 2003.

[46] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[47] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. Academic Press, London, 1982.

[48] G. Owen. *Game Theory*. Academic Press, New York, NY, 1982.

[49] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. Annual Symposium on Foundations of Computer Science*, pages 132–142, 1978.

[50] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic systhesis of fine-motion strategies for robots. *Int. J. Robot. Res.*, 3(1):3–24, 1984.

[51] B. R. Donald. *Error Detection and Recovery for Robot Motion Planning with Uncertainty*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1987.

[52] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Trans. Robot. & Autom.*, 4(4):369–379, August 1988.

[53] R. H. Taylor, M. T. Mason, and K. Y. Goldberg. Sensor-based manipulation planning as a game with nature. In *Fourth International Symposium on Robotics Research*, pages 421–429, 1987.

[54] K. Y. Goldberg and M. T. Mason. Bayesian grasping. In *IEEE Int. Conf. Robot. & Autom.*, 1990.

[55] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason. Sensorless parts feeding with a one joint robot. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 229–237. A K Peters, Wellesley, MA, 1997.

[56] V. J. Lumelsky and A. A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[57] I. Kamon, E. Rivlin, and E. Rimon. Range-sensor based navigation in three dimensions. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[58] B. Tovar, S. M. LaValle, and R. Murrieta. Optimal navigation and object finding without geometric maps or localization. In *Proc. IEEE International Conference on Robotics and Automation*, pages 464–470, 2003.

[59] L. Guilamo, B. Tovar, and S. M. LaValle. Pursuit-evasion in an unknown environment using gap navigation graphs. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[60] J. Y. Herve', P. Cucka, and R. Sharma. Qualitative visual control of a robot manipulator. In *Image Understanding Workshop*, pages 895–908, 1990.

[61] B. R. Donald and J. Jennings. Sensor interpretation and task-directed planning using perceptual equivalence classes. In *IEEE Int. Conf. Robot. & Autom.*, pages 190–197, Sacramento, CA, April 1991.

[62] B. R. Donald. On information invariants in robotics. *Artif. Intell.*, 72:217–304, 1995.

[63] H. Choset and J. Burdick. Sensor based planning, part I: The generalized Voronoi graph. In *IEEE Int. Conf. Robot. & Autom.*, pages 1649–1655, 1995.

[64] A. Stentz. Optimal and efficient path planning for partially-known environments. In *IEEE Int. Conf. Robot. & Autom.*, pages 3310–3317, 1994.

[65] T. D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Application of Graphs*, pages 426–441. Springer-Verlag, Berlin, 1976.

[66] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM J. Computing*, 21(5):863–888, October 1992.

[67] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. *International Journal of Computational Geometry and Applications*, 9(5):471–494, 1999.

[68] S.-M. Park, J.-H. Lee, and K.-Y. Chwa. Visibility-based pursuit-evasion in a polygonal region by a searcher. Technical Report CS/TR-2001-161, KAIST, Dept. of Computer Science, Korea, January 2001.

[69] Y. Yavin and M. Pachter. *Pursuit-Evasion Differential Games.* Pergamon Press, Oxford, England, 1987.

[70] R. Simmons and S. Koenig. Probabilistic navigation in partially observable environments. In *Proc. Int. Joint Conf. on Artif. Intell.*, 1995.

[71] D. Fox, W. Burgard, S. Thrun, and A. B. Cremers. Position estimation for mobile robots in dynamic environments. In *Proc. Am. Assoc. Artif. Intell.*, 1998.

[72] D. J. Austin and P. Jensfelt. Using multiple gaussian hypotheses to represent probability distributions for mobile robot localization. In *IEEE Int. Conf. Robot. & Autom.*, pages 1036–1041, 2000.

[73] G. Dudek, K. Romanik, and S. Whitesides. Localizing a robot with minimum travel. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 437–446, 1995.

[74] M. Rao, G. Dudek, and S. Whitesides. Randomized algorithms for minimum distance localization. In *Proc. Workshop on Algorithmic Foundations of Robotics*, pages 265–280, 2004.

[75] Jason M. O'Kane and Steven M. LaValle. Almost-sensorless localization. In *IEEE Int. Conf. Robot. & Autom.*, 2005.

[76] E. Remolina and B. Kuipers. A logical account of causal and topological maps. In *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 5–11, 2001.

[77] M. Littman L. Kaelbling and and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[78] N. Zhang and W. Lin. A model approximation scheme for planning in partially observable stochastic domains. *Journal of Artificial Intelligence Research*, 7:199–230, 1997.

[79] M. Hauskrecht. Value functions approximations for partially observable markov decision processes. *JAIR*, 13:33–94, 1994.

[80] M. Hauskrecht. Incremental methods for computing bounds in partially observable markov decision processes. In *14th National Conference on Artificial Intelligence*, Providence, Rhode Island, 1997. AAAI Press / MIT Press.

[81] J. G. Hocking and G. S. Young. *Topology*. Dover, 1988.

[82] H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems*. Wiley, New York, NY, 1972.

[83] D. Fox, S. Thrun, W. Burgard, and F. Dellaert. Particle filters for mobile robot localization. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pages 470–498, New York, 2001. Springer.

[84] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.

[85] S. Akella and M. Mason. Using partial sensor information to orient parts. *IJRR*, 18:963–997, 1999.

[86] M. Ghallib, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[87] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 2nd Edition*. Pearson Education, Inc., Upper Saddle River, NJ, 2003.

[88] G. Della Riccia, D. Dubois, R. Kruse, and H-J. Lenz, editors. *Planning Based on Decision Theory*. Springer-Verlag, Vienna, 2003.

[89] P. D. Straffin. *Game Theory and Strategy.* Mathematical Assoc. of America, Washington, D.C., 1993.