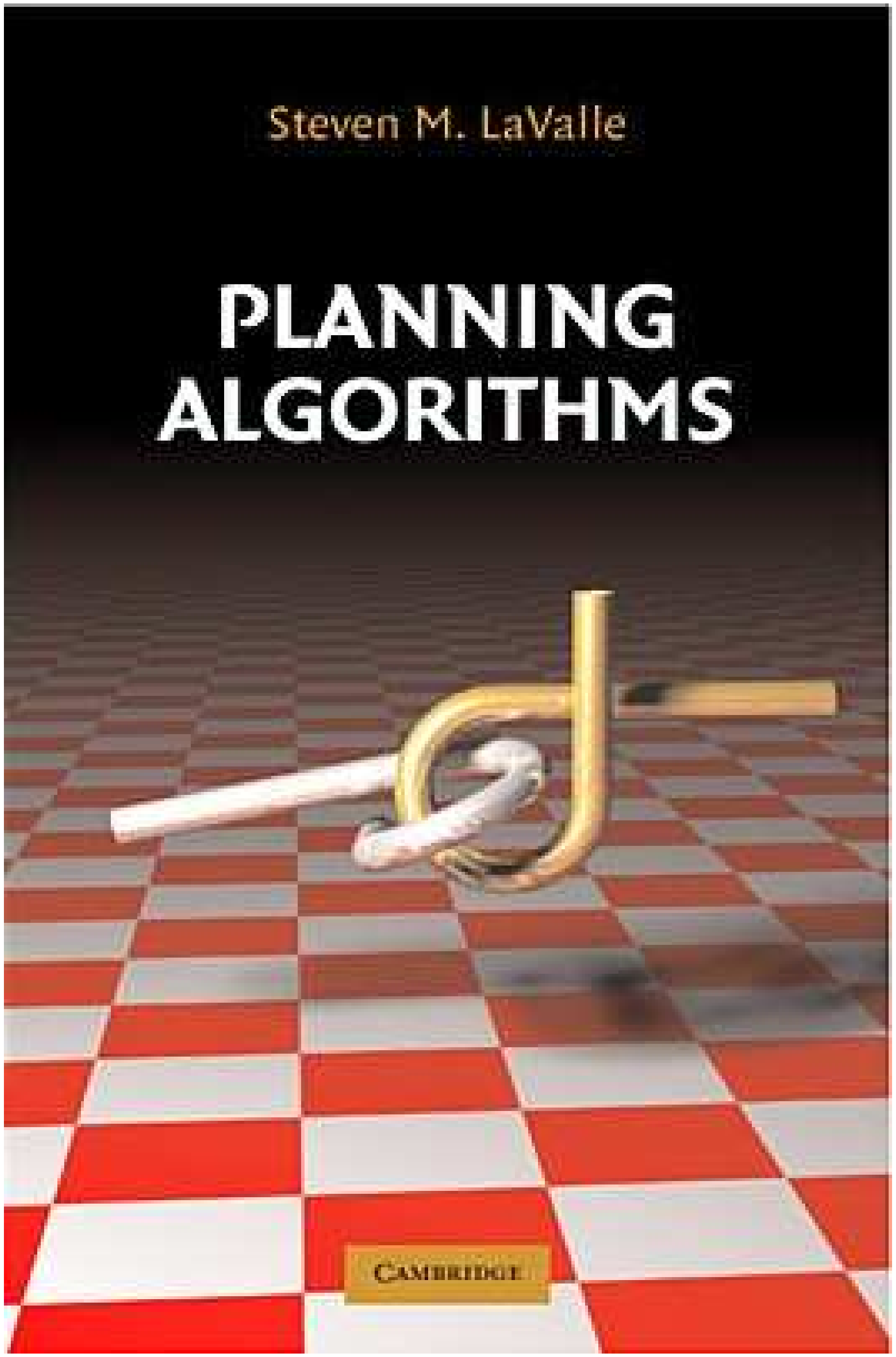


Part I

Introductory Material

Steven M. LaValle

University of Illinois

Available for downloading at **http://planning.cs.uiuc.edu/**

Published by Cambridge University Press

Chapter 1

Introduction

1.1 Planning to Plan

Planning is a term that means different things to different groups of people. *Robotics* addresses the automation of mechanical systems that have sensing, actuation, and computation capabilities (similar terms, such as *autonomous systems* are also used). A fundamental need in robotics is to have algorithms that convert high-level specifications of tasks from humans into low-level descriptions of how to move. The terms *motion planning* and *trajectory planning* are often used for these kinds of problems. A classical version of motion planning is sometimes referred to as the *Piano Mover's Problem*. Imagine giving a precise computer-aided design (CAD) model of a house and a piano as input to an algorithm. The algorithm must determine how to move the piano from one room to another in the house without hitting anything. Most of us have encountered similar problems when moving a sofa or mattress up a set of stairs. Robot motion planning usually ignores dynamics and other differential constraints and focuses primarily on the translations and rotations required to move the piano. Recent work, however, does consider other aspects, such as uncertainties, differential constraints, modeling errors, and optimality. Trajectory planning usually refers to the problem of taking the solution from a robot motion planning algorithm and determining how to move along the solution in a way that respects the mechanical limitations of the robot.

Control theory has historically been concerned with designing inputs to physical systems described by differential equations. These could include mechanical systems such as cars or aircraft, electrical systems such as noise filters, or even systems arising in areas as diverse as chemistry, economics, and sociology. Classically, control theory has developed *feedback policies*, which enable an adaptive response during execution, and has focused on *stability*, which ensures that the dynamics do not cause the system to become wildly out of control. A large emphasis is also placed on optimizing criteria to minimize resource consumption, such as energy or time. In recent control theory literature, *motion planning* sometimes refers to

the construction of inputs to a nonlinear dynamical system that drives it from an initial state to a specified goal state. For example, imagine trying to operate a remote-controlled hovercraft that glides over the surface of a frozen pond. Suppose we would like the hovercraft to leave its current resting location and come to rest at another specified location. Can an algorithm be designed that computes the desired inputs, even in an ideal simulator that neglects uncertainties that arise from model inaccuracies? It is possible to add other considerations, such as uncertainties, feedback, and optimality; however, the problem is already challenging enough without these.

In *artificial intelligence*, the terms *planning* and *AI planning* take on a more discrete flavor. Instead of moving a piano through a continuous space, as in the robot motion planning problem, the task might be to solve a puzzle, such as the Rubik's cube or a sliding-tile puzzle, or to achieve a task that is modeled discretely, such as building a stack of blocks. Although such problems could be modeled with continuous spaces, it seems natural to define a finite set of actions that can be applied to a discrete set of states and to construct a solution by giving the appropriate sequence of actions. Historically, planning has been considered different from *problem solving*; however, the distinction seems to have faded away in recent years. In this book, we do not attempt to make a distinction between the two. Also, substantial effort has been devoted to representation language issues in planning. Although some of this will be covered, it is mainly outside of our focus. Many decision-theoretic ideas have recently been incorporated into the AI planning problem, to model uncertainties, adversarial scenarios, and optimization. These issues are important and are considered in detail in Part III.

Given the broad range of problems to which the term planning has been applied in the artificial intelligence, control theory, and robotics communities, you might wonder whether it has a specific meaning. Otherwise, just about anything could be considered as an instance of planning. Some common elements for planning problems will be discussed shortly, but first we consider planning as a branch of algorithms. Hence, this book is entitled *Planning Algorithms*. The primary focus is on algorithmic and computational issues of planning problems that have arisen in several disciplines. On the other hand, this does not mean that planning algorithms refers to an existing community of researchers within the general algorithms community. This book it not limited to combinatorics and asymptotic complexity analysis, which is the main focus in pure algorithms. The focus here includes numerous concepts that are not necessarily algorithmic but aid in modeling, solving, and analyzing planning problems.

Natural questions at this point are, What is a plan? How is a plan represented? How is it computed? What is it supposed to achieve? How is its quality evaluated? Who or what is going to use it? This chapter provides general answers to these questions. Regarding the user of the plan, it clearly depends on the application. In most applications, an algorithm executes the plan; however, the user could even be a human. Imagine, for example, that the planning algorithm provides you with

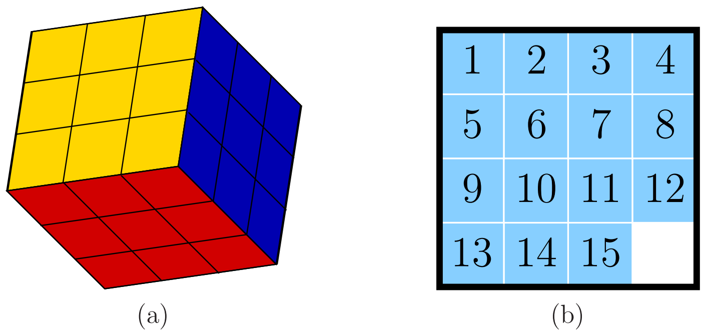


Figure 1.1: The Rubik's cube (a), sliding-tile puzzle (b), and other related puzzles are examples of discrete planning problems.

an investment strategy.

In this book, the user of the plan will frequently be referred to as a *robot* or a *decision maker*. In artificial intelligence and related areas, it has become popular in recent years to use the term *agent*, possibly with adjectives to yield an *intelligent agent* or *software agent*. Control theory usually refers to the decision maker as a *controller*. The plan in this context is sometimes referred to as a *policy* or *control law*. In a game-theoretic context, it might make sense to refer to decision makers as *players*. Regardless of the terminology used in a particular discipline, this book is concerned with planning algorithms that find a strategy for one or more decision makers. Therefore, remember that terms such as *robot*, *agent*, and *controller* are interchangeable.

1.2 Motivational Examples and Applications

Planning problems abound. This section surveys several examples and applications to inspire you to read further.

Why study planning algorithms? There are at least two good reasons. First, it is fun to try to get machines to solve problems for which even humans have great difficulty. This involves exciting challenges in modeling planning problems, designing efficient algorithms, and developing robust implementations. Second, planning algorithms have achieved widespread successes in several industries and academic disciplines, including robotics, manufacturing, drug design, and aerospace applications. The rapid growth in recent years indicates that many more fascinating applications may be on the horizon. These are exciting times to study planning algorithms and contribute to their development and use.

Discrete puzzles, operations, and scheduling Chapter 2 covers discrete planning, which can be applied to solve familiar puzzles, such as those shown in

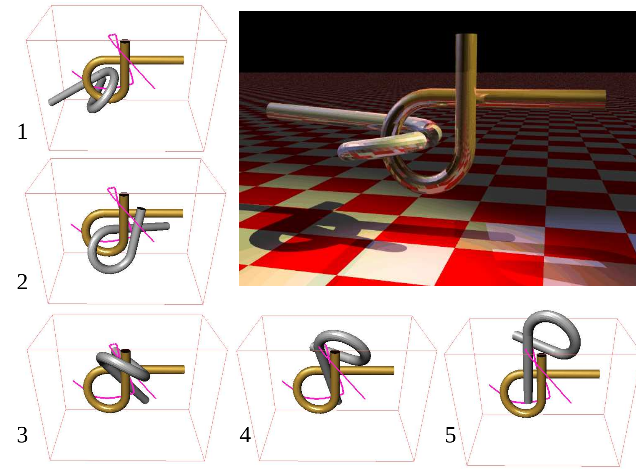


Figure 1.2: Remember puzzles like this? Imagine trying to solve one with an algorithm. The goal is to pull the two bars apart. This example is called the Alpha 1.0 Puzzle. It was created by Boris Yamrom and posted as a research benchmark by Nancy Amato at Texas A&M University. This solution and animation were made by James Kuffner (see [38] for the full movie).

Figure 1.1. They are also good at games such as chess or bridge [58]. Discrete planning techniques have been used in space applications, including a rover that traveled on Mars and the Earth Observing One satellite [11, 23, 57]. When combined with methods for planning in continuous spaces, they can solve complicated tasks such as determining how to bend sheet metal into complicated objects [25]; see Section 7.5 for the related problem of folding cartons.

A motion planning puzzle The puzzles in Figure 1.1 can be easily discretized because of the regularity and symmetries involved in moving the parts. Figure 1.2 shows a problem that lacks these properties and requires planning in a continuous space. Such problems are solved by using the motion planning techniques of Part II. This puzzle was designed to frustrate both humans and motion planning algorithms. It can be solved in a few minutes on a standard personal computer (PC) using the techniques in Section 5.5. Many other puzzles have been developed as benchmarks for evaluating planning algorithms.

An automotive assembly puzzle Although the problem in Figure 1.2 may appear to be pure fun and games, similar problems arise in important applications. For example, Figure 1.3 shows an automotive assembly problem for which software is needed to determine whether a wiper motor can be inserted (and removed)

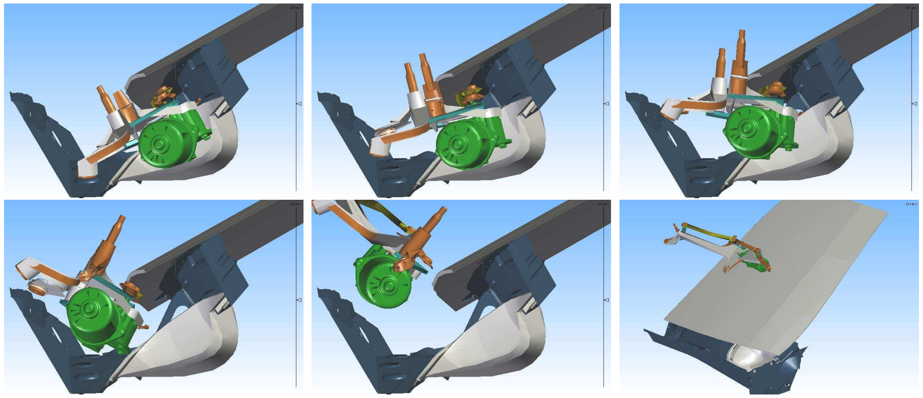


Figure 1.3: An automotive assembly task that involves inserting or removing a windshield wiper motor from a car body cavity. This problem was solved for clients using the motion planning software of Kineo CAM (courtesy of Kineo CAM).

from the car body cavity. Traditionally, such a problem is solved by constructing physical models. This costly and time-consuming part of the design process can be virtually eliminated in software by directly manipulating the CAD models.

The wiper example is just one of many. The most widespread impact on industry comes from motion planning software developed at Kineo CAM. It has been integrated into Robcad (eM-Workplace) from Tecnomatix, which is a leading tool for designing robotic workcells in numerous factories around the world. Their software has also been applied to assembly problems by Renault, Ford, Airbus, Optivus, and many other major corporations. Other companies and institutions are also heavily involved in developing and delivering motion planning tools for industry (many are secret projects, which unfortunately cannot be described here). One of the first instances of motion planning applied to real assembly problems is documented in [9].

Sealing cracks in automotive assembly Figure 1.4 shows a simulation of robots performing sealing at the Volvo Cars assembly plant in Torslanda, Sweden. Sealing is the process of using robots to spray a sticky substance along the seams of a car body to prevent dirt and water from entering and causing corrosion. The entire robot workcell is designed using CAD tools, which automatically provide the necessary geometric models for motion planning software. The solution shown in Figure 1.4 is one of many problems solved for Volvo Cars and others using motion planning software developed by the Fraunhofer Chalmers Centre (FCC). Using motion planning software, engineers need only specify the high-level task of performing the sealing, and the robot motions are computed automatically. This saves enormous time and expense in the manufacturing process.

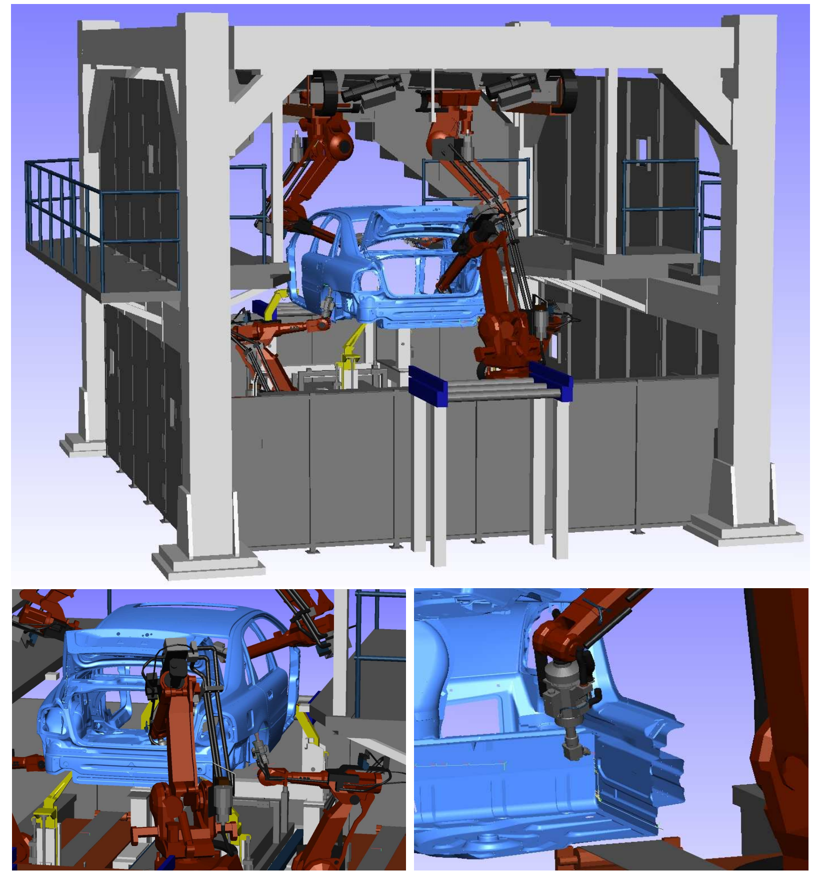


Figure 1.4: An application of motion planning to the sealing process in automotive manufacturing. Planning software developed by the Fraunhofer Chalmers Centre (FCC) is used at the Volvo Cars plant in Sweden (courtesy of Volvo Cars and FCC).

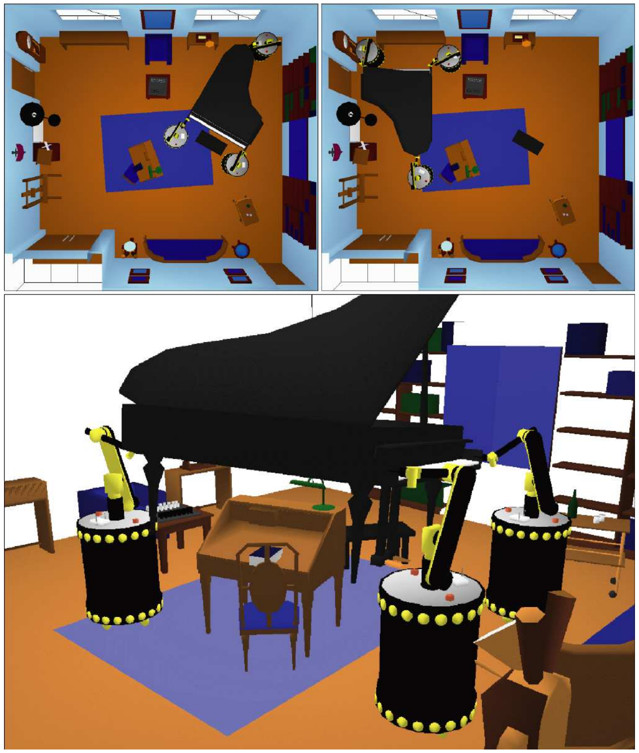
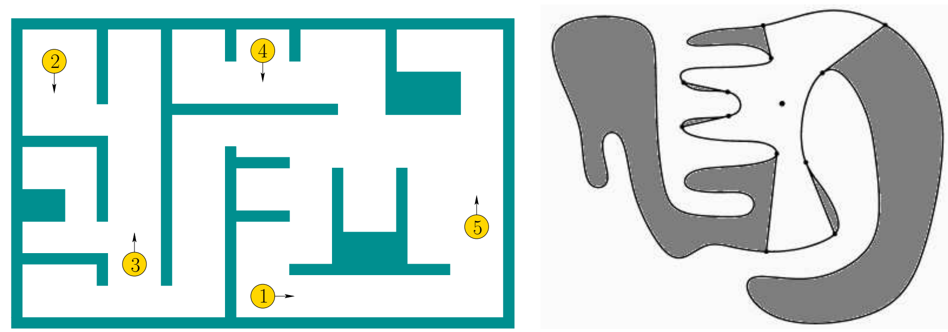


Figure 1.5: Using mobile robots to move a piano [13].

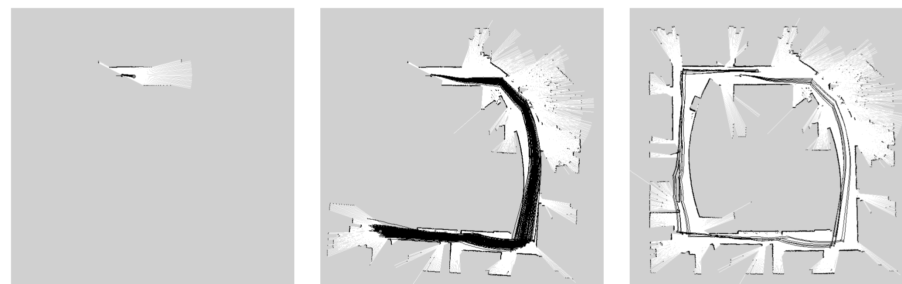
Moving furniture Returning to pure entertainment, the problem shown in Figure 1.5 involves moving a grand piano across a room using three mobile robots with manipulation arms mounted on them. The problem is humorously inspired by the phrase *Piano Mover's Problem*. Collisions between robots and with other pieces of furniture must be avoided. The problem is further complicated because the robots, piano, and floor form closed kinematic chains, which are covered in Sections 4.4 and 7.4.

Navigating mobile robots A more common task for mobile robots is to request them to navigate in an indoor environment, as shown in Figure 1.6a. A robot might be asked to perform tasks such as building a map of the environment, determining its precise location within a map, or arriving at a particular place. Acquiring and manipulating information from sensors is quite challenging and is covered in Chapters 11 and 12. Most robots operate in spite of large uncertainties. At one extreme, it may appear that having many sensors is beneficial because it could allow precise estimation of the environment and the robot po-

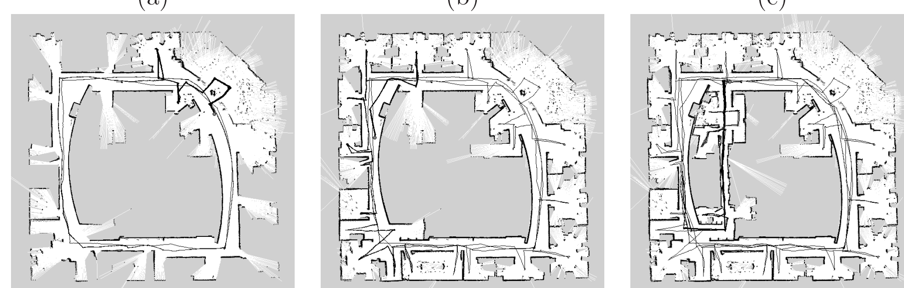


(a) (b)

Figure 1.6: (a) Several mobile robots attempt to successfully navigate in an indoor environment while avoiding collisions with the walls and each other. (b) Imagine using a lantern to search a cave for missing people.



(a) (b) (c)



(d) (e) (f)

Figure 1.7: A mobile robot can reliably construct a good map of its environment (here, the Intel Research Lab) while simultaneously localizing itself. This is accomplished using laser scanning sensors and performing efficient Bayesian computations on the information space [20].

sition and orientation. This is the premise of many existing systems, as shown for the robot system in Figure 1.7, which constructs a map of its environment. It may alternatively be preferable to develop low-cost and reliable robots that achieve specific tasks with little or no sensing. These trade-offs are carefully considered in Chapters 11 and 12. Planning under uncertainty is the focus of Part III.

If there are multiple robots, then many additional issues arise. How can the robots communicate? How can their information be integrated? Should their coordination be centralized or distributed? How can collisions between them be avoided? Do they each achieve independent tasks, or are they required to collaborate in some way? If they are competing in some way, then concepts from game theory may apply. Therefore, some game theory appears in Sections 9.3, 9.4, 10.5, 11.7, and 13.5.

Playing hide and seek One important task for a mobile robot is playing the game of hide and seek. Imagine entering a cave in complete darkness. You are given a lantern and asked to search for any people who might be moving about, as shown in Figure 1.6b. Several questions might come to mind. Does a strategy even exist that guarantees I will find everyone? If not, then how many other searchers are needed before this task can be completed? Where should I move next? Can I keep from exploring the same places multiple times? This scenario arises in many robotics applications. The robots can be embedded in surveillance systems that use mobile robots with various types of sensors (motion, thermal, cameras, etc.). In scenarios that involve multiple robots with little or no communication, the strategy could help one robot locate others. One robot could even try to locate another that is malfunctioning. Outside of robotics, software tools can be developed that assist people in systematically searching or covering complicated environments, for applications such as law enforcement, search and rescue, toxic cleanup, and in the architectural design of secure buildings. The problem is extremely difficult because the status of the pursuit must be carefully computed to avoid unnecessarily allowing the evader to sneak back to places already searched. The information-space concepts of Chapter 11 become critical in solving the problem. For an algorithmic solution to the hide-and-seek game, see Section 12.4.

Making smart video game characters The problem in Figure 1.6b might remind you of a video game. In the arcade classic *Pacman*, the ghosts are programmed to seek the player. Modern video games involve human-like characters that exhibit much more sophisticated behavior. Planning algorithms can enable game developers to program character behaviors at a higher level, with the expectation that the character can determine on its own how to move in an intelligent way.

At present there is a large separation between the planning-algorithm and

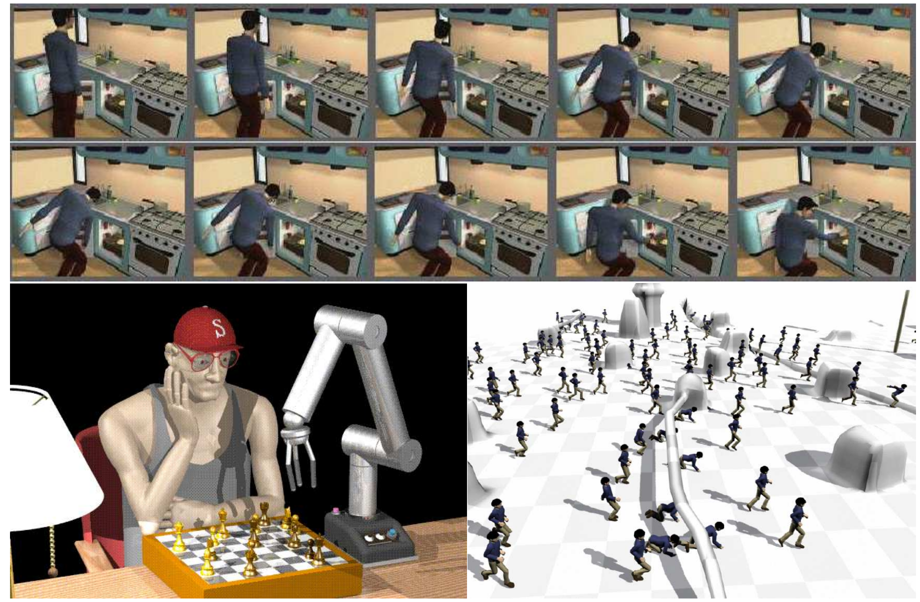
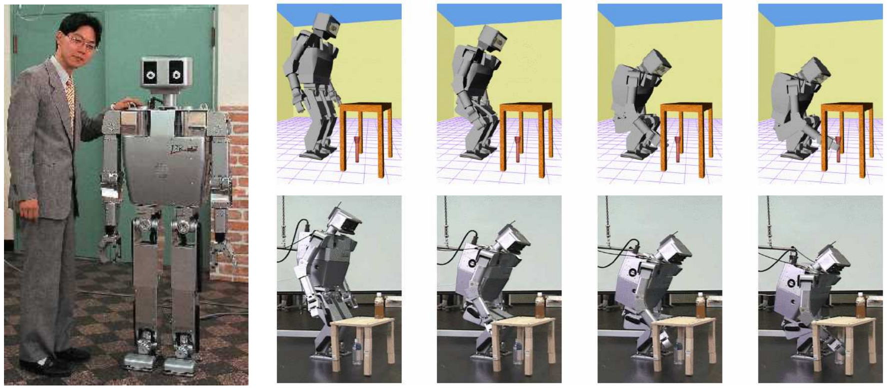


Figure 1.8: Across the top, a motion computed by a planning algorithm, for a digital actor to reach into a refrigerator [32]. In the lower left, a digital actor plays chess with a virtual robot [35]. In the lower right, a planning algorithm computes the motions of 100 digital actors moving across terrain with obstacles [41].

video-game communities. Some developers of planning algorithms are recently considering more of the particular concerns that are important in video games. Video-game developers have to invest too much energy at present to adapt existing techniques to their problems. For recent books that are geared for game developers, see [6, 21].

Virtual humans and humanoid robots Beyond video games, there is broader interest in developing virtual humans. See Figure 1.8. In the field of computer graphics, computer-generated animations are a primary focus. Animators would like to develop digital actors that maintain many elusive style characteristics of human actors while at the same time being able to design motions for them from high-level descriptions. It is extremely tedious and time consuming to specify all motions frame-by-frame. The development of planning algorithms in this context is rapidly expanding.

Why stop at *virtual* humans? The Japanese robotics community has inspired the world with its development of advanced humanoid robots. In 1997, Honda shocked the world by unveiling an impressive humanoid that could walk up stairs

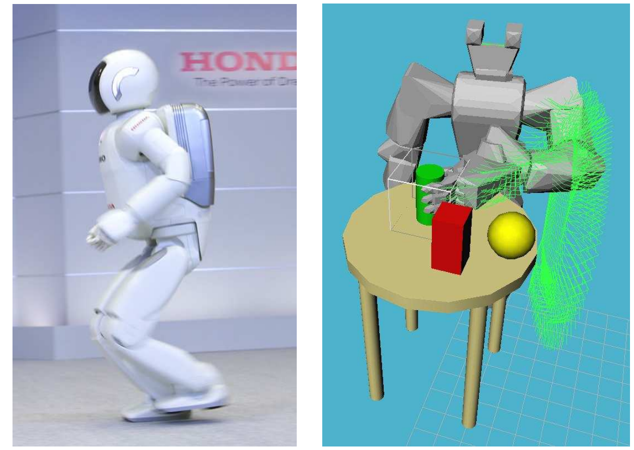


(a) (b)

Figure 1.9: (a) This is a picture of the H7 humanoid robot and one of its developers, S. Kagami. It was developed in the JSK Laboratory at the University of Tokyo. (b) Bringing virtual reality and physical reality together. A planning algorithm computes stable motions for a humanoid to grab an obstructed object on the floor [39].

and recover from lost balance. Since that time, numerous corporations and institutions have improved humanoid designs. Although most of the mechanical issues have been worked out, two principle difficulties that remain are sensing and planning. What good is a humanoid robot if it cannot be programmed to accept high-level commands and execute them autonomously? Figure 1.9 shows work from the University of Tokyo for which a plan computed in simulation for a humanoid robot is actually applied on a real humanoid. Figure 1.10 shows humanoid projects from the Japanese automotive industry.

Parking cars and trailers The planning problems discussed so far have not involved differential constraints, which are the main focus in Part IV. Consider the problem of parking slow-moving vehicles, as shown in Figure 1.11. Most people have a little difficulty with parallel parking a car and much greater difficulty parking a truck with a trailer. Imagine the difficulty of parallel parking an airport baggage train! See Chapter 13 for many related examples. What makes these problems so challenging? A car is constrained to move in the direction that the rear wheels are pointing. Maneuvering the car around obstacles therefore becomes challenging. If all four wheels could turn to any orientation, this problem would vanish. The term *nonholonomic planning* encompasses parking problems and many others. Figure 1.12a shows a humorous driving problem. Figure 1.12b shows an extremely complicated vehicle for which nonholonomic planning algorithms were developed and applied in industry.

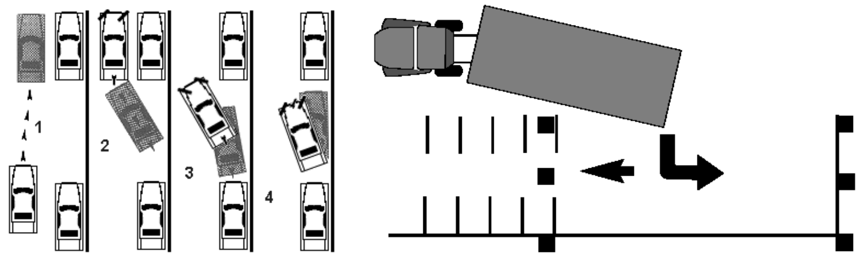


(a) (b)

Figure 1.10: Humanoid robots from the Japanese automotive industry: (a) The latest Asimo robot from Honda can run at 3 km/hr (courtesy of Honda); (b) planning is incorporated with vision in the Toyota humanoid so that it plans to grasp objects [27].

"Wreckless" driving Now consider driving the car at high speeds. As the speed increases, the car must be treated as a dynamical system due to momentum. The car is no longer able to instantaneously start and stop, which was reasonable for parking problems. Although there exist planning algorithms that address such issues, there are still many unsolved research problems. The impact on industry has not yet reached the level achieved by ordinary motion planning, as shown in Figures 1.3 and 1.4. By considering dynamics in the design process, performance and safety evaluations can be performed before constructing the vehicle. Figure 1.13 shows a solution computed by a planning algorithm that determines how to steer a car at high speeds through a town while avoiding collisions with buildings. A planning algorithm could even be used to assess whether a sports utility vehicle tumbles sideways when stopping too quickly. Tremendous time and costs can be spared by determining design flaws early in the development process via simulations and planning. One related problem is *verification*, in which a mechanical system design must be thoroughly tested to make sure that it performs as expected in spite of all possible problems that could go wrong during its use. Planning algorithms can also help in this process. For example, the algorithm can try to violently crash a vehicle, thereby establishing that a better design is needed.

Aside from aiding in the design process, planning algorithms that consider dynamics can be directly embedded into robotic systems. Figure 1.13b shows an



(a) (b)

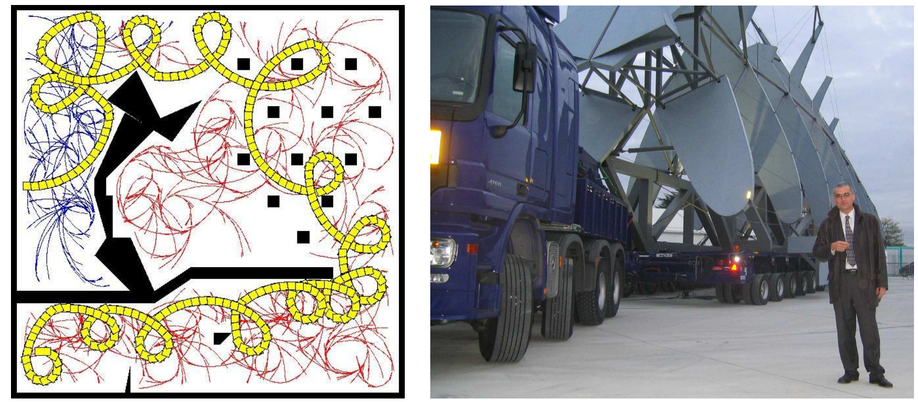
Figure 1.11: Some parking illustrations from government manuals for driver testing: (a) parking a car (from the 2005 *Missouri Driver Guide*); (b) parking a tractor trailer (published by the Pennsylvania Division of Motor Vehicles). Both humans and planning algorithms can solve these problems.

application that involves a difficult combination of most of the issues mentioned so far. Driving across rugged, unknown terrain at high speeds involves dynamics, uncertainties, and obstacle avoidance. Numerous unsolved research problems remain in this context.

Flying Through the Air or in Space Driving naturally leads to flying. Planning algorithms can help to navigate autonomous helicopters through obstacles. They can also compute thrusts for a spacecraft so that collisions are avoided around a complicated structure, such as a space station. In Section 14.1.3, the problem of designing entry trajectories for a reusable spacecraft is described. Mission planning for interplanetary spacecraft, including solar sails, can even be performed using planning algorithms [26].

Designing better drugs Planning algorithms are even impacting fields as far away from robotics as computational biology. Two major problems are protein folding and drug design. In both cases, scientists attempt to explain behaviors in organisms by the way large organic molecules interact. Such molecules are generally flexible. Drug molecules are small (see Figure 1.14), and proteins usually have thousands of atoms. The *docking problem* involves determining whether a flexible molecule can insert itself into a protein cavity, as shown in Figure 1.14, while satisfying other constraints, such as maintaining low energy. Once geometric models are applied to molecules, the problem looks very similar to the assembly problem in Figure 1.3 and can be solved by motion planning algorithms. See Section 7.5 and the literature at the end of Chapter 7.

Perspective Planning algorithms have been applied to many more problems than those shown here. In some cases, the work has progressed from modeling,



(a) (b)

Figure 1.12: (a) Having a little fun with differential constraints. An obstacle-avoiding path is shown for a car that must move forward and can only turn left. Could you have found such a solution on your own? This is an easy problem for several planning algorithms. (b) This gigantic truck was designed to transport portions of the Airbus A380 across France. Kineo CAM developed nonholonomic planning software that plans routes through villages that avoid obstacles and satisfy differential constraints imposed by 20 steering axles. Jean-Paul Laumond, a pioneer of nonholonomic planning, is also pictured.

(a) (b)

Figure 1.13: Reckless driving: (a) Using a planning algorithm to drive a car quickly through an obstacle course [10]. (b) A contender developed by the Red Team from Carnegie Mellon University in the DARPA Grand Challenge for autonomous vehicles driving at high speeds over rugged terrain (courtesy of the Red Team).

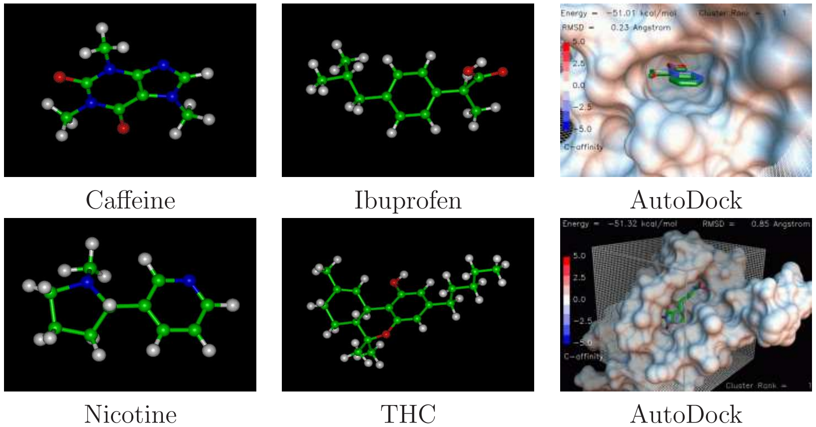


Figure 1.14: On the left, several familiar drugs are pictured as ball-and-stick models (courtesy of the New York University MathMol Library [44]). On the right, 3D models of protein-ligand docking are shown from the AutoDock software package (courtesy of the Scripps Research Institute).

to theoretical algorithms, to practical software that is used in industry. In other cases, substantial research remains to bring planning methods to their full potential. The future holds tremendous excitement for those who participate in the development and application of planning algorithms.

1.3 Basic Ingredients of Planning

Although the subject of this book spans a broad class of models and problems, there are several basic ingredients that arise throughout virtually all of the topics covered as part of planning.

State Planning problems involve a *state space* that captures all possible situations that could arise. The *state* could, for example, represent the position and orientation of a robot, the locations of tiles in a puzzle, or the position and velocity of a helicopter. Both discrete (finite, or countably infinite) and continuous (uncountably infinite) state spaces will be allowed. One recurring theme is that the state space is usually represented *implicitly* by a planning algorithm. In most applications, the size of the state space (in terms of number of states or combinatorial complexity) is much too large to be explicitly represented. Nevertheless, the definition of the state space is an important component in the formulation of a planning problem and in the design and analysis of algorithms that solve it.

Time All planning problems involve a sequence of decisions that must be applied over time. Time might be explicitly modeled, as in a problem such as driving a car as quickly as possible through an obstacle course. Alternatively, time may be implicit, by simply reflecting the fact that actions must follow in succession, as in the case of solving the Rubik's cube. The particular time is unimportant, but the proper sequence must be maintained. Another example of implicit time is a solution to the Piano Mover's Problem; the solution to moving the piano may be converted into an animation over time, but the particular speed is not specified in the plan. As in the case of state spaces, time may be either discrete or continuous. In the latter case, imagine that a continuum of decisions is being made by a plan.

Actions A plan generates *actions* that manipulate the state. The terms *actions* and *operators* are common in artificial intelligence; in control theory and robotics, the related terms are *inputs* and *controls*. Somewhere in the planning formulation, it must be specified how the state changes when actions are applied. This may be expressed as a state-valued function for the case of discrete time or as an ordinary differential equation for continuous time. For most motion planning problems, explicit reference to time is avoided by directly specifying a path through a continuous state space. Such paths could be obtained as the integral of differential equations, but this is not necessary. For some problems, actions could be chosen by *nature*, which interfere with the outcome and are not under the control of the decision maker. This enables uncertainty in predictability to be introduced into the planning problem; see Chapter 10.

Initial and goal states A planning problem usually involves starting in some initial state and trying to arrive at a specified goal state or any state in a set of goal states. The actions are selected in a way that tries to make this happen.

A criterion This encodes the desired outcome of a plan in terms of the state and actions that are executed. There are generally two different kinds of planning concerns based on the type of criterion:

1. **Feasibility:** Find a plan that causes arrival at a goal state, regardless of its efficiency.

2. **Optimality:** Find a feasible plan that optimizes performance in some carefully specified manner, in addition to arriving in a goal state.

For most of the problems considered in this book, feasibility is already challenging enough; achieving optimality is considerably harder for most problems. Therefore, much of the focus is on finding feasible solutions to problems, as opposed to optimal solutions. The majority of literature in robotics, control theory, and related fields focuses on optimality, but this is not necessarily important for many problems of interest. In many applications, it is difficult to even formulate the right criterion to optimize. Even if a desirable criterion can be formulated, it may be impossible to obtain a practical algorithm that computes optimal plans. In such cases, feasible solutions are certainly preferable to having no solutions at all. Fortunately, for many algorithms the solutions produced are not too far from optimal in practice. This reduces some of the motivation for finding optimal solutions. For problems that involve probabilistic uncertainty, however, optimization arises more frequently. The probabilities are often utilized to obtain the best performance in terms of expected costs. Feasibility is often associated with performing a worst-case analysis of uncertainties.

A plan In general, a plan imposes a specific strategy or behavior on a decision maker. A plan may simply specify a sequence of actions to be taken; however, it could be more complicated. If it is impossible to predict future states, then the plan can specify actions as a function of state. In this case, regardless of the future states, the appropriate action is determined. Using terminology from other fields, this enables *feedback* or *reactive plans*. It might even be the case that the state cannot be measured. In this case, the appropriate action must be determined from whatever information is available up to the current time. This will generally be referred to as an *information state*, on which the actions of a plan are conditioned.

1.4 Algorithms, Planners, and Plans

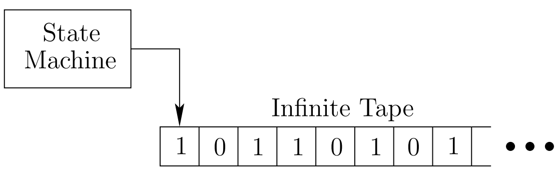


Figure 1.15: According to the Church-Turing thesis, the notion of an algorithm is equivalent to the notion of a Turing machine.

1.4.1 Algorithms

What is a planning algorithm? This is a difficult question, and a precise mathematical definition will not be given in this book. Instead, the general idea will be explained, along with many examples of planning algorithms. A more basic question is, What is an algorithm? One answer is the classical Turing machine model, which is used to define an algorithm in theoretical computer science. A *Turing machine* is a finite state machine with a special head that can read and

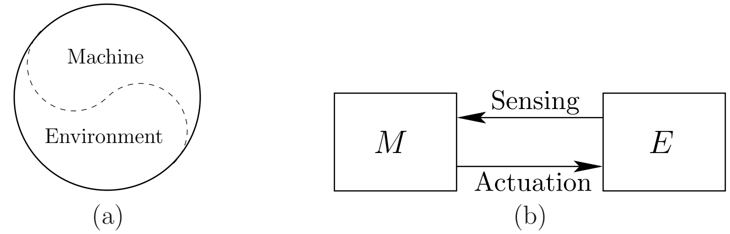


Figure 1.16: (a) The boundary between machine and environment is considered as an arbitrary line that may be drawn in many ways depending on the context. (b) Once the boundary has been drawn, it is assumed that the machine, M, interacts with the environment, E, through sensing and actuation.

write along an infinite piece of tape, as depicted in Figure 1.15. The Church-Turing thesis states that an algorithm *is* a Turing machine (see [29, 55] for more details). The *input* to the algorithm is encoded as a string of symbols (usually a binary string) and then is written to the tape. The Turing machine reads the string, performs computations, and then decides whether to *accept* or *reject* the string. This version of the Turing machine only solves *decision problems*; however, there are straightforward extensions that can yield other desired outputs, such as a plan.

The Turing model is reasonable for many of the algorithms in this book; however, others may not exactly fit. The trouble with using the Turing machine in some situations is that plans often interact with the physical world. As indicated in Figure 1.16, the boundary between the machine and the environment is an arbitrary line that varies from problem to problem. Once drawn, *sensors* provide information about the environment; this provides input to the machine during execution. The machine then executes actions, which provides *actuation* to the environment. The actuation may alter the environment in some way that is later measured by sensors. Therefore, the machine and its environment are closely coupled during execution. This is fundamental to robotics and many other fields in which planning is used.

Using the Turing machine as a foundation for algorithms usually implies that the physical world must be first carefully modeled and written on the tape before the algorithm can make decisions. If changes occur in the world during execution of the algorithm, then it is not clear what should happen. For example, a mobile robot could be moving in a cluttered environment in which people are walking around. As another example, a robot might throw an object onto a table without being able to precisely predict how the object will come to rest. It can take measurements of the results with sensors, but it again becomes a difficult task to determine how much information should be explicitly modeled and written on the tape. The *on-line algorithm* model is more appropriate for these kinds of problems

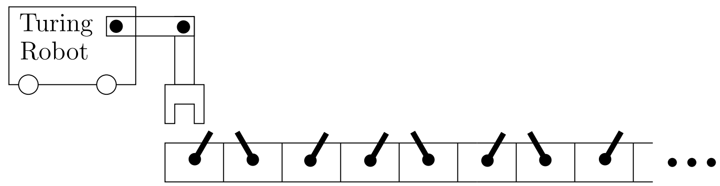


Figure 1.17: A robot and an infinite sequence of switches could be used to simulate a Turing machine. Through manipulation, however, many other kinds of behavior could be obtained that fall outside of the Turing model.

[33, 48, 56]; however, it still does not capture a notion of algorithms that is broad enough for all of the topics of this book.

Processes that occur in a physical world are more complicated than the interaction between a state machine and a piece of tape filled with symbols. It is even possible to simulate the tape by imagining a robot that interacts with a long row of switches as depicted in Figure 1.17. The switches serve the same purpose as the tape, and the robot carries a computer that can simulate the finite state machine.[1] The complicated interaction allowed between a robot and its environment could give rise to many other models of computation.[2] Thus, the term *algorithm* will be used somewhat less formally than in the theory of computation. Both *planners* and *plans* are considered as algorithms in this book.

1.4.2 Planners

A planner simply constructs a plan and may be a machine or a human. If the planner is a machine, it will generally be considered as a planning algorithm. In many circumstances it is an algorithm in the strict Turing sense; however, this is not necessary. In some cases, humans become planners by developing a plan that works in all situations. For example, it is perfectly acceptable for a human to design a state machine that is connected to the environment (see Section 12.3.1). There are no additional inputs in this case because the human fulfills the role of the algorithm. The planning model is given as input to the human, and the human "computes" a plan.

1.4.3 Plans

Once a plan is determined, there are three ways to use it:

[1]Of course, having infinitely long tape seems impossible in the physical world. Other versions of Turing machines exist in which the tape is finite but as long as necessary to process the given input. This may be more appropriate for the discussion.

[2]Performing computations with mechanical systems is discussed in [52]. Computation models over the reals are covered in [5].

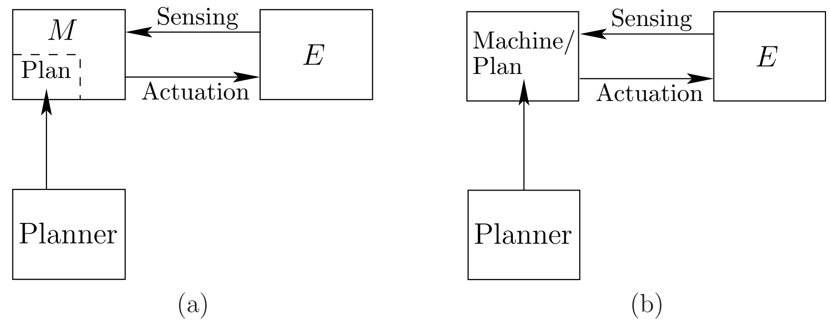


Figure 1.18: (a) A planner produces a plan that may be executed by the machine. The planner may either be a machine itself or even a human. (b) Alternatively, the planner may design the entire machine.

1. **Execution:** Execute it either in simulation or in a mechanical device (robot) connected to the physical world.

2. **Refinement:** Refine it into a better plan.

3. **Hierarchical Inclusion:** Package it as an action in a higher level plan.

Each of these will be explained in succession.

Execution A plan is usually executed by a machine. A human could alternatively execute it; however, the case of machine execution is the primary focus of this book. There are two general types of machine execution. The first is depicted in Figure 1.18a, in which the planner produces a *plan*, which is encoded in some way and given as input to the machine. In this case, the machine is considered *programmable* and can accept possible plans from a planner before execution. It will generally be assumed that once the plan is given, the machine becomes autonomous and can no longer interact with the planner. Of course, this model could be extended to allow machines to be improved over time by receiving better plans; however, we want a strict notion of autonomy for the discussion of planning in this book. This approach does not prohibit the updating of plans in practice; however, this is not preferred because plans should already be designed to take into account new information during execution.

The second type of machine execution of a plan is depicted in Figure 1.18b. In this case, the plan produced by the planner encodes an entire machine. The plan *is* a special-purpose machine that is designed to solve the specific tasks given originally to the planner. Under this interpretation, one may be a *minimalist* and design the simplest machine possible that sufficiently solves the desired tasks. If the plan is encoded as a finite state machine, then it can sometimes be considered

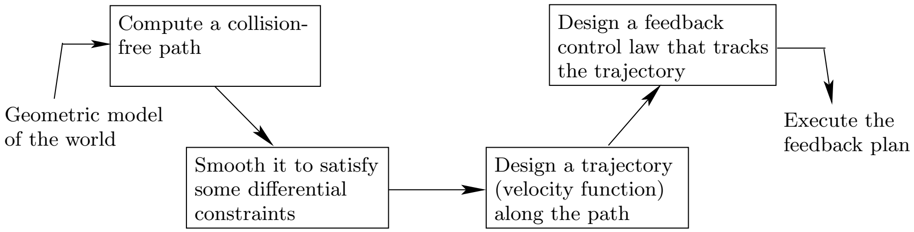


Figure 1.19: A refinement approach that has been used for decades in robotics.

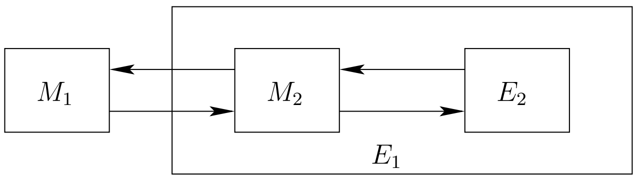


Figure 1.20: In a hierarchical model, the environment of one machine may itself contain a machine.

as an algorithm in the Turing sense (depending on whether connecting the machine to a tape preserves its operation).

Refinement If a plan is used for refinement, then a planner accepts it as input and determines a new plan that is hopefully an improvement. The new plan may take more problem aspects into account, or it may simply be more efficient. Refinement may be applied repeatedly, to produce a sequence of improved plans, until the final one is executed. Figure 1.19 shows a refinement approach used in robotics. Consider, for example, moving an indoor mobile robot. The first plan yields a collision-free path through the building. The second plan transforms the route into one that satisfies differential constraints based on wheel motions (recall Figure 1.11). The third plan considers how to move the robot along the path at various speeds while satisfying momentum considerations. The fourth plan incorporates feedback to ensure that the robot stays as close as possible to the planned path in spite of unpredictable behavior. Further elaboration on this approach and its trade-offs appears in Section 14.6.1.

Hierarchical inclusion Under hierarchical inclusion, a plan is incorporated as an action in a larger plan. The original plan can be imagined as a subroutine in the larger plan. For this to succeed, it is important for the original plan to guarantee *termination*, so that the larger plan can execute more actions as needed. Hierarchical inclusion can be performed any number of times, resulting in a rooted *tree* of plans. This leads to a general model of *hierarchical planning*. Each vertex in the tree is a plan. The root vertex represents the *master plan*. The children

of any vertex are plans that are incorporated as actions in the plan of the vertex. There is no limit to the tree depth or number of children per vertex. In hierarchical planning, the line between machine and environment is drawn in multiple places. For example, the environment, E_1, with respect to a machine, M_1, might actually include another machine, M_2, that interacts with its environment, E_2, as depicted in Figure 1.20. Examples of hierarchical planning appear in Sections 7.3.2 and 12.5.1.

1.5 Organization of the Book

Here is a brief overview of the book. See also the overviews at the beginning of Parts II–IV.

PART I: Introductory Material

This provides very basic background for the rest of the book.

- **Chapter 1: Introductory Material**
  This chapter offers some general perspective and includes some motivational examples and applications of planning algorithms.

- **Chapter 2: Discrete Planning**
  This chapter covers the simplest form of planning and can be considered as a springboard for entering into the rest of the book. From here, you can continue to Part II, or even head straight to Part III. Sections 2.1 and 2.2 are most important for heading into Part II. For Part III, Section 2.3 is additionally useful.

PART II: Motion Planning

The main source of inspiration for the problems and algorithms covered in this part is robotics. The methods, however, are general enough for use in other applications in other areas, such as computational biology, computer-aided design, and computer graphics. An alternative title that more accurately reflects the kind of planning that occurs is "Planning in Continuous State Spaces."

- **Chapter 3: Geometric Representations and Transformations**
  The chapter gives important background for expressing a motion planning problem. Section 3.1 describes how to construct geometric models, and the remaining sections indicate how to transform them. Sections 3.1 and 3.2 are important for later chapters.

- **Chapter 4: The Configuration Space**
  This chapter introduces concepts from topology and uses them to formulate the *configuration space*, which is the state space that arises in motion planning. Sections 4.1, 4.2, and 4.3.1 are important for understanding most of the material in later chapters. In addition to the previously mentioned

sections, all of Section 4.3 provides useful background for the combinatorial methods of Chapter 6.

- **Chapter 5: Sampling-Based Motion Planning**
  This chapter introduces motion planning algorithms that have dominated the literature in recent years and have been applied in fields both in and out of robotics. If you understand the basic idea that the configuration space represents a continuous state space, most of the concepts should be understandable. They even apply to other problems in which continuous state spaces emerge, in addition to motion planning and robotics. Chapter 14 revisits sampling-based planning, but under differential constraints.

- **Chapter 6: Combinatorial Motion Planning**
  The algorithms covered in this section are sometimes called *exact algorithms* because they build discrete representations without losing any information. They are *complete*, which means that they must find a solution if one exists; otherwise, they report failure. The sampling-based algorithms have been more useful in practice, but they only achieve weaker notions of completeness.

- **Chapter 7: Extensions of Basic Motion Planning**
  This chapter introduces many problems and algorithms that are extensions of the methods from Chapters 5 and 6. Most can be followed with basic understanding of the material from these chapters. Section 7.4 covers planning for closed kinematic chains; this requires an understanding of the additional material, from Section 4.4

- **Chapter 8: Feedback Motion Planning**
  This is a transitional chapter that introduces feedback into the motion planning problem but still does not introduce differential constraints, which are deferred until Part IV. The previous chapters of Part II focused on computing *open-loop* plans, which means that any errors that might occur during execution of the plan are ignored, yet the plan will be executed as planned. Using feedback yields a *closed-loop* plan that responds to unpredictable events during execution.

PART III: Decision-Theoretic Planning

An alternative title to Part III is "Planning Under Uncertainty." Most of Part III addresses discrete state spaces, which can be studied immediately following Part I. However, some sections cover extensions to continuous spaces; to understand these parts, it will be helpful to have read some of Part II.

- **Chapter 9: Basic Decision Theory**
  The main idea in this chapter is to design the best decision for a decision maker that is confronted with interference from other decision makers. The others may be true opponents in a game or may be fictitious in order to model uncertainties. The chapter focuses on making a decision in a single step and provides a building block for Part III because planning under uncertainty can be considered as multi-step decision making.

- **Chapter 10: Sequential Decision Theory**
  This chapter takes the concepts from Chapter 9 and extends them by chaining together a sequence of basic decision-making problems. Dynamic programming concepts from Section 2.3 become important here. For all of the problems in this chapter, it is assumed that the current state is always known. All uncertainties that exist are with respect to prediction of future states, as opposed to measuring the current state.

- **Chapter 11: Sensors and Information Spaces**
  The chapter extends the formulations of Chapter 10 into a framework for planning when the current state is unknown during execution. Information regarding the state is obtained from sensor observations and the memory of actions that were previously applied. The information space serves a similar purpose for problems with sensing uncertainty as the configuration space has for motion planning.

- **Chapter 12: Planning Under Sensing Uncertainty**
  This chapter covers several planning problems and algorithms that involve sensing uncertainty. This includes problems such as localization, map building, pursuit-evasion, and manipulation. All of these problems are unified under the idea of planning in information spaces, which follows from Chapter 11.

PART IV: Planning Under Differential Constraints

This can be considered as a continuation of Part II. Here there can be both global (obstacles) and local (differential) constraints on the continuous state spaces that arise in motion planning. Dynamical systems are also considered, which yields state spaces that include both position and velocity information (this coincides with the notion of a *state space* in control theory or a *phase space* in physics and differential equations).

- **Chapter 13: Differential Models**
  This chapter serves as an introduction to Part IV by introducing numerous models that involve differential constraints. This includes constraints that arise from wheels rolling as well as some that arise from the dynamics of mechanical systems.

- **Chapter 14: Sampling-Based Planning Under Differential Constraints**
  Algorithms for solving planning problems under the models of Chapter 13

are presented. Many algorithms are extensions of methods from Chapter 5. All methods are sampling-based because very little can be accomplished with combinatorial techniques in the context of differential constraints.

- **Chapter 15: System Theory and Analytical Techniques**
  This chapter provides an overview of the concepts and tools developed mainly in control theory literature. They are complementary to the algorithms of Chapter 14 and often provide important insights or components in the development of planning algorithms under differential constraints.