1 Rationally Engineering Rational Robots

Leslie Pack Kaelbling and Tomás Lozano-Pérez

Engineered structure is critical for embodied intelligence

1.1 Introduction

Our ultimate goal is to design and build general-purpose robots with human-level task and domain generality, including the ability to learn whole new competencies online. We would like our robots to be able to do anything a human can do, including serve as general household helpers, hospital aides, or disaster relief workers.

Our particular focus is on the computer software that provides all of the control and decision-making for the robot. A robot controller consumes a stream of input signals from cameras, joint-angle sensors, microphones, etc., and produces a stream of output signals to control the robot's joints, as well as natural-language utterances to humans or signals to other robots.

The software has to enable a robot to live a long lifetime in a complex world: it should be competent at performing the basic tasks of locomotion and manipulation, be capable of taking instructions, and adapt to changes in its environment, ranging from low-level calibration changes to human owners with very different habits and requirements.

We are faced with two tightly intertwined driving questions:

1. How should this software be structured?
2. What process should we, as engineers, use to develop this software?

Answering these questions is an enormous challenge. We address them through an engineering design approach, ultimately concluding that ***modularity, compositionality, and model-based run-time rationality*** **are foundational design principles that will enable an effective process for engineering generally intelligent robots.**

1.1.1 Engineering Intelligent Robots

Ultimately, we want our robots to perform well, on average, over all possible situations in which they might find themselves, over a time horizon of days or

weeks. We can express this formally by saying that we want to find a policy π, in the form of a mapping of the history of all the inputs the robot has ever received, as well as actions it has ever generated, into the next action to take, that maximizes

$$\sum_{w \in W} V(\pi, w) P(w) \ .$$

This is the sum *over all possible worlds w that the robot could encounter* of the *value V* of executing π in w. So, the engineering specification consists of $P(w)$, the distribution over possible worlds, and V, a description of how much we *value* the results of executing π in world w. A possible world includes the particular state of the world, the objective for the robot in that world, and the laws governing the world dynamics.[1]

So, our question is, what form should π have, and how should we construct it? We describe this process as happening "in the factory:" that is, before the robots are deployed into the domains where they will be working (such as peoples' houses, or hospitals, or shopping malls). There's a spectrum of approaches to obtaining programs for general-purpose intelligent robots.

At one extreme (*design*), we humans engineer the system as much as possible, based on human insight and experience, exploiting everything we know about the world distribution $P(w)$, including knowledge of math, physics, and psychology. The systems we engineer must also include state-estimation and machine-learning algorithms to be used "on the job"—when the robot is actually deployed—to enable it to learn about and adapt to its particular world. Pragmatically, the pure design strategy aligns with the classical practice in robotics of breaking the problem down and engineering solutions to each subproblem. This strategy has been highly effective in complex problems, such as space missions and nuclear reactors, where the input signals are relatively simple and the programmers understand the "possible worlds" in detail. But humans don't know, at a sufficiently explicit level, how to write the programs necessary for general processing and interpretation of input images or for controlling even relatively simple robots over the range of possible worlds that, e.g. a household robot, would face.

At the other extreme (*evolution*), we humans construct a complex simulation that, in some sense, covers the distribution $P(w)$ of possible worlds a robot could be deployed in. Coupled with this simulation, we use a completely general-purpose learning algorithm, with as little built-in bias about the domain

1. This view can encompass other agents, as long as we can model them as having (possibly unknown) fixed stochastic strategies, but does not extend to a full game-theoretic treatment of other agents.

as possible, to find—in the factory—a policy π that performs well over all the simulated domains, on average. If the variability in the distribution over possible worlds is such that the robot will have to adapt or learn on the job, then it will be up to the process of learning in the factory to invent any necessary data structures or learning algorithms needed to accomplish that adaptation.

Our Design Criteria

There are many reasonable positions one could take along the spectrum between the pure design and pure evolution approaches. How should we decide between them? Our primary design criteria are:

1. The amount of engineering work required (including human-design of aspects of the solution π as well as setting up simulations, gathering training data, etc.).
2. The amount of computation (or learning time in "playground" physical environments) required in the factory.
3. The general performance of the robot during deployment, including, for example, the amount of training required to learn new abilities during deployment.
4. The deployment-time computational efficiency of the robot program.

Additional criteria of real importance include the ease with which humans can debug, understand, and explain the robot's behavior (in the factory and during deployment) and the ability to make safety guarantees.

Current Approaches

Many current approaches aspire to minimize the human-design aspect (as advocated by Sutton (2019)), to avoid the imposition of incorrect biases on the system. Some of these approaches use reinforcement-learning algorithms in simulation to learn policies for behaving in the world. Other current approaches attempt to bypass the difficulties of reinforcement-learning systems, notably exploration, by focusing on supervised learning from large-scale imitation data, whether from teleoperation or internet videos.

In either case, there is still a *substantial* amount of human design involved: selecting input and output spaces, designing reward functions and learning algorithms, choosing the distribution of training environments, picking hyperparameters, etc. These methods have been surprisingly successful at relatively limited-scope problems, but they have not led to solutions for broad-scope behavior outside of games, in large part because the size of the neural networks that would be required to represent very general-purpose policies is very large,

and the amount of real or simulated experience required to learn them is enormous. Furthermore, very few of these approaches allow the robot to acquire new abilities during its deployment. Importantly, however, these methods have been able to synthesize policies that substantially outperform more strongly engineered methods for tasks such as locomotion and dexterous manipulation. Importantly, for this class of tasks, the learned sensori-motor policies are very efficient to execute during deployment, making them capable of relatively high-frequency control.

Despite the effectiveness of current approaches for learning sensorimotor-control policies, it remains unclear to us how these methods can scale to the kind of highly general-purpose capacity for intelligent behavior that we are aiming for, without some additional structure.

1.1.2 Generality and its Discontents

Why is it that implementing or learning very general-purpose policies is so difficult? Fundamentally, it is because of an enormous growth in the number of possible courses of action the robot might have to take, as the size and variability of the domain increases.

Environmental Complexity

Robots must interact with an enormous variety of objects, with a rich variation in 3D shape, kinematic structure, mass properties, material composition, and appearance. Even basic grasping presents enormous challenges when viewed in generality. The 3D shape of the object interacts with the shape of the gripper to determine grasp stability; small surface details of the object can affect the quality of contact; the material of both object and gripper affects friction and how much pressure can be applied; the mass distribution affects which grasps are stable; the object size determines whether two arms are needed; the presence of another object in a gripper might require the use of other body surfaces; and so on.

Importantly, the required robot actions depend not only on the objects directly relevant to the current task, such as the tabletop to be cleaned, but also on a wide array of other objects, including objects on the tabletop, chairs near the table, etc. Putting a sugar box in the pantry might require re-organizing a shelf to make room. Cutting boa a cucumber might require removing a wrapper and finding an appropriate knife and cutting board.

Furthermore, the whole robot shape interacts with the environment; it is not just the grippers that matter. So, all the furniture near a tabletop to be cleaned constrains the robot motions; any objects occluding the doors to the pantry must be cleared out; any pots on top of the cutting board must be

moved. We require a policy that can handle any combination of objects in any configuration.

Task Horizon

The notion of *horizon* has many definitions, but the one we find most relevant is, informally, the number of primitive actions in the future that one must consider to choose the current action. So, for example, a problem in which the robot must take a lot of actions, but each next action is obvious given the current state, has a short horizon. But a problem in which the robot must take several actions in order to enable later actions, or take care not to expend all its resources too soon, has a longer horizon.

Generally we can structure behavior hierarchically, so that the horizon is limited at every behavioral level. When driving to a destination, selecting a route has a long horizon (even at an abstract level where actions are relatively limited); but given the next intersection to aim for, the lower-level walking or driving behavior can have a relatively short horizon.

General manipulation tasks, in which object interactions are central, tend to not decouple as readily and have more complex hierarchical dependencies. It is generally not possible to construct an abstract plan that ignores object properties and robot capabilities. Putting a box of sugar away might require considering motions for many objects with interacting constraints, including not putting objects down in front of the pantry door before opening it. The size and weight of an object might require getting a wheelbarrow, which will constrain the future path. As a result, manipulation tasks tend to be relatively long horizon.

Partial observability

Most robotics task are *partially observable*: the robot lacks complete knowledge of the state of the world; it has only inherently partial and noisy observations. Much current work in robotics assumes that images sufficiently define the current world state, so that one can learn a policy that maps the current image (or a few recent images) to actions. This assumption rules out a vast swath of applications. Partial observability comes in many forms: object shape self-occlusion from a single view, occlusion from other objects (e. g. the drawer where the knife is stored), mass and material properties not evident from a camera view, complete lack of knowledge of the objects in the next room, uncertainty about the goals and behavior of humans in the room, etc.

The crucial point here is that the appropriate robot actions at any moment in time depend not just on the robot's current sensory observations. In general, the actions should depend on the history of observations of the robot (or a

summary in the form of a "belief state") as well as general knowledge (priors) about the world.

Combinatorial complexity

Approaches to learning policies that attempt to limit designed structure must cope with the combination of environmental complexity, task horizon and partial observability required for competent robot behavior over a wide range of possible worlds. Although such learning is possible in principle, it would require intractable amounts of diverse experience, whether in the form of simulation or "internet scale" data and unfathomable amounts of computation.

1.1.3 Modular, Compositional, Rational Systems

Our contention is that we can exploit what human designers know to break down the problem complexity, structuring a system design to exploit what humans are good at while using machine-learning methods to efficiently learn the rest.

Modularity

One important step is to design a system that is *modular*, in that it is made up of multiple components that are *independently* learnable. Training multiple smaller modules can require exponentially less data than training one large one. So, just knowing something about the modular substructure of a function can substantially reduce the data requirements for learning it.

We advocate building systems with modularity at multiple different scales. Coarse-grained modularity appears in the overall "architecture" of the system, perhaps including modules specialized for perception, memory, and planning, as well as large pre-trained language, vision-language, or video models, as illustrated in Section 1.3.2.

Compositionality

Modules are even more powerful if they are not fixed into a static assembly, but are more fine-grained, and can be recombined flexibly to address new problems. The idea arises in natural-language semantics, articulated by Barbara Partee: "The meaning of a compound expression is a function of the meanings of its parts and of the way they are syntactically combined." (Partee 1984) Formal languages like first-order logic and computer programming languages also, by design, have compositional semantics and rich, flexible structures that often allow a very complex computation to have a very simple programmatic description.

Compositional systems enable positive *combinatorial explosions*: a small set of syntactic components can combine to make infinitely many sentences; a small set of concepts can combine to make infinitely many complex meanings; a small set of primitive actions can be combined to make infinitely many plans to solve infinitely many problems.

One critical advantage of compositional systems of this kind is that they can be learned independently because the individual components have separate semantics. For example, an individual "neuron" in a typical neural network has meaning in terms of the role it plays in the larger system, but it is difficult to assign local, individual semantics to it. However, the meaning of the word "teacup" can be learned largely independently of the meanings of other words, and son can be reused in a wide variety of situations.

Rationality

Given a compositional system for generating candidate action sequences and for predicting and evaluating their effects, the robot can *plan* to act *rationally*; i.e., choose actions that maximize its expected future rewards. Explicitly reasoning at deployment time to choose actions is slower than executing a pre-learned feedforward policy, but it avoids an enormous amount of work in the factory, to derive a policy that has, effectively, already solved all the problems that it could potentially ever face. It is difficult to imagine this approach being feasible for addressing the enormous variability of the human world.

Systems that carry out some form of on-line inference can be proved to be more powerful and more efficient to learn than pure feedforward policies (C.-C. Lin et al. 2021). A great deal of empirical evidence also supports this conclusion (Brown and Sandholm 2019).

Modular, compositional, rational systems in robotics

What opportunities do we have for modular design in a robot policy? At the coarse-grained level, we can assemble a collection of engineered or pre-trained models with individual competences, ranging from classic motor controllers and kinematics modules to modern large language and vision models. In the designs we have in mind, some of these larger modules will have a more fine-grained compositional structure which will be exploited by search-based planning and inference algorithms. We outline some of the primary opportunities for compositional subsystems here and will return to them in more detail later.

- If we have controllers that can drive the robot to achieve some relatively local and short-term "subgoals," (grasping an object, moving safely to a target pose, and placing) then we can recombine them in many ways to

solve longer-term problems, either using a higher-level control program that decides which controllers to call or using search-based planning algorithms to compose them into goal-achieving sequences.
- If we have causal or observational "facts" about the robot's world (dropping an egg tends to break it, milk is usually kept in the fridge, etc.) we can combine them at runtime using inference and planning algorithms to draw conclusions that are implied (logically and/or statistically) by the facts.
- If we have modules that can evaluate whether constraints hold in the world (the robot would collide with the table if it stood over there, or the plate is sufficiently upright that the sandwich won't slip off, for example), then we can combine them into tests for combinations (the robot won't collide and the sandwich won't slip off), allowing an inference algorithm to try to achieve the combined objective.
- Relatedly, if we know a description for the set of solutions to a subproblem (such as motion trajectories) that satisfy one condition and a description for the set of solutions that satisfy another, we can combine these (via set intersection) to directly find solutions that satisfy both.

1.1.4 Meeting Our Design Criteria

We believe that one (not the only!) way to design generally intelligent robots that meets our specifications is for engineers to specify basic modular decomposition including multiple compositional systems, and then to perform learning in multiple stages, both inside the factory and on the job.

- This will be not *too* hard for the engineers, who can use familiar design and specification methods.
- It will be *dramatically* more computationally efficient to learn many small modules that are flexibly recomposed to solve problems.
- Because the individual modules have independent and clear world semantics, new perceptual detectors, controllers, and constraints can be *learned on the job* and added to the system without causing that forgetting of previous knowledge, which can happen with monolithic learned models.
- Because the system's models are in a combinatorial set of modules, it can require more computation at inference (deployment, prediction) time to generate behavioral decisions. This problem can be mitigated through "caching" solutions to common problem instances, effectively producing a partial policy that covers many common cases efficiently, but still retaining generality.

In the following sections, we describe an overall design and implementation strategy, propose a concrete coarse-grained cognitive architecture, provide

more detail about compositional learning and inference for one of these modules (task and motion planning), and finish with a discussion of meta-cognitive strategies for deciding how to allocate online computation.

1.2 Phases of Design And Implementation

Our approach to designing and implementing rational robots operates in three phases:

1. *Mind design*: using a combination of the study of naturally intelligent systems, the fundamental structure of the physical world, the mathematics of learning and reasoning, and the competences of existing large models, determine a coarse-grained modular decomposition of the robot system.
2. *Implementation in the factory*: using a combination of classical methods and machine learning techniques in simulation and from large-scale data, engineer and learn general-purpose models and modules that will serve as the basis for robots that can be "shipped" to real deployments (for example, in homes or hospitals).
3. *Learning on the job*: using lifelong, active learning methods, including learning from instruction, demonstration and exploration, a robot adapts itself to the particular environment it finds itself in (for example, tuning its process for the common requests made of it, learning the household's organizational style, and acquiring specific new skills and vocabulary).

We can make a weak analogy between these phases and phases in the development of a natural intelligence. Very coarsely, we can see mind-design phase as corresponding to evolution, the in-the-factory learning phase as corresponding to early childhood development, and the on-the-job learning phase as corresponding to learning as an adult. Let's consider each phase in turn.

1.2.1 Mind Design

Humans engineers are best at specifying algorithms and structural properties of a domain, and relatively poor at specifying detailed parameter values or even procedures for carrying out common operations. At this level, we study general properties of our environment and of new-born natural systems, to try to find human-expressible fundamental principles of intelligent behavior.

An important source of inspiration here are natural systems. Many cognitive scientists, such as Spelke and Kinzler (2007), believe that humans (and many non-human animals) are born with fundamental mechanisms for reasoning about their physical and social world, including an understanding of solid, permanent objects, the fundamental relations in three-dimensional space, and the idea that there are other agents in the world.

These insights inform our design of data structures: abstracting over objects, building computational map representations isomorphic to 2D or 3D space, and constructing convolutional or recurrent networks as appropriate for the problems at hand. In Section 1.3.1 we elaborate a particular set of design choices in more detail.

1.2.2 Learning in the Factory

Even after engineers specify some invariant aspects of the domain and the overall structure of the robot's computation, there is a lot to do! Because we are not pursuing a completely end-to-end approach, we think instead of particular sub-modules that might, to some degree, be mixed and matched to make a deployable robot system. Some modules will be specific to particular robot task distributions (a factory robot will require different competences than a space exploration robot) or morphology, but others will be more generic.

- Perception: segmenting an image into a set of physically coherent objects; estimating their mass and material; integrating observations over time to understand 3D shape; making connections to human language.
- Physics: approximate simulation of physical dynamics based on initial image or object arrangements, including evaluation of stability, etc.
- Navigation: policies for short-horizon navigation through a visible obstacle field; reaching around and among obstacles for local-space manipulation.
- Control: policies for walking on uneven terrain, robust grasping with tactile feedback, inserting objects into tight spaces, in-hand manipulation, etc.
- Cultural knowledge: How should one behave in a restaurant? Is the present King of France bald? What is the word for "crow" in Finnish? What is a good recipe for chocolate cake?
- Human interaction: What are good ways of getting people to work together? How can I interpret someone's navigational intent?

While some of these basic competences are available in the form of pre-existing engineered modules (such as physics simulators or inverse kinematics modules), many will require learning, ranging from tuning of existing methods to acquisition of completely novel models.

Many of these competences are becoming available, with more or less generality and accuracy, in the form of so-called "foundation models," including large-language models (OpenAI 2024a), video-prediction models (OpenAI 2024b), object segmentation models (Kirillov et al. 2023) and local navigation models (Shah et al. 2023).

In Section 1.5 we explore a particular flexible representation and set of learning strategies for robot behavior that can serve as the "glue" for combining these general competences into a specific robot system.

1.2.3 Learning on the Job

When a robot is sent from the factory to a new job, it should be highly competent in a generic sense (able to navigate, manipulate objects robustly, carry on a basic conversation, remember what happened yesterday, etc.) but may not know the specifics that it needs for its new job. These specifics could include new vocabulary and associated visual detectors, new motor skills, and new high-level processes as well as specific facts about this world: who the people are, how the house is arranged, where the whisks are kept, etc. Critically, these are compositional elements that can be added to the existing competences.

Assume that the robot is always engaged in behaving in the world: it receives goals from a human or other external process and makes and executes plans to solve them using its current set of operations. During its "free time," it might choose to experiment with objects that are available to it, to improve its skills (Kumar et al. 2024).

When a job arises that the robot does not know how to perform, such as tightening a bolt with a wrench, a human will help the robot add a new operation to its repertoire. It is important to note that the addition and generalization of the operation happens incrementally, and the robot will continue to improve, refine, and generalize its ability to execute the operation and its model of its own capabilities as it continues to carry out the jobs it is assigned.

The human might begin by teaching the robot to recognize new "concepts" to add to its vocabulary for high-level planning and reasoning, and then demonstrates a way to achieve them. This work will have an enormous payoff, because it will allow the robot to use the learned operations in a wide variety of combinations and situations to address novel problems in future.

The problem setting of learning on the job has unique constraints and opportunities. It is made difficult by the requirement of being able to learn immediately from just a few demonstrations and a relatively small amount of self-supervised training. Offsetting this difficulty is the fact that it need only operate in the circumstances it finds itself in: initially, it is fine to learn to operate just the one particular wrench in the shop or to set the table using only the everyday blue dishes. Generalization from these initial situations can happen over time, as the robot gains more experience; but the robot will be able to perform competently in familiar situations with very little training.

1.3 Mind Design for Rational Robots

Our primary commitment in designing the robot's mind is to *model-based rationality*. That is, that the robot should build up, based on its history of perception and actions, an explicit representation of a *belief* about the external state of the world. It should then use flexible, goal-directed planning and inference mechanisms to combine its current belief about the world state with a "world model" that characterizes the effects of its actions, to select actions that maximizes its expected utility. We argue that this approach is *necessary* for achieving the generality and flexibility required of an intelligent robot with feasible computational resources.

1.3.1 Compositionality and Rationality

We can see rational inference as a general-purpose method for converting information from one form into another. In our context, the available information consists of perceptual observation histories, which we must map into an action. Most simply and directly, we could use an explicit *feed-forward policy*, in the form of a control law or a look-up table or a neural network. However, there are many other representational choices, including *value functions* (mappings from the current state and an action choice into their long-term utility) and *world models* (mappings from a state and action into a resulting state), together with an explicit goal or reward function. This is a pivotal design choice.

Both value functions and world models are *implicit* policy representations. A value function can be understood as a type of *energy-based model*, which requires an optimization process to select the next action, given a current state. A world model is even more indirect, requiring a possibly computationally expensive planning search process to select the next action.

We adopt the approach of learning world models and using them to infer, given a reward or goal function, what actions to take, using the *principle of rationality*, which states that a rational agent should select actions that will maximize its expected future utility, in expectation given its current information. This approach to implicit policy representation can be modeled as one of sequential decision-making under uncertainty.

Why would we consider using a representation that requires online computational work to determine an action? There are several important reasons.

- *Sample complexity*: For many problems, there is a clear sense in which the world model is a simpler object than the policy that it implicitly encodes. Interestingly, recent results in learning theory (C.-C. Lin et al. 2021) show that in some cases representations that exploit online computation can learn

richer function classes from the same amount of data than pure feed-forward or auto-regressive models.

In addition, because world models characterize the dynamics of the external world, and the external world is highly structured with locality and sparsity of effect, world models can be made highly compositional. We can represent the effects of different actions in separate modules; we can represent the dynamics of different aspects of the world state in separate modules. These modules compose so as to further reduce learning complexity because the parameters in the individual modules are re-used whenever the modules can be dynamically recombined to solve new problems.

Finally, world models can be learned via *self-supervision*: every action the robot takes in the world provides a training example for predicting the results of its actions. Supervised learning methods, which are generally much more sample-efficient than policy or value learning, can be applied directly to these data to acquire world models.

- *Goal-independence*: A world model is a *declarative* specification of how the world works, independent of any particular objective. It can be used, via inference, to determine behavior for *any possible goal*. By contrast, policies and value functions are generally goal-specific; although it is possible to condition them on a representation of a goal, this conditioning dramatically increases their complexity and generally can only address a substantially restricted space of goals.
- *Understandability*: For complex problems, many factors may contribute to a decision about which action to take next, involving optimization over many aspects of the past, present, and future world states. World and reward models have more clearly understandable local semantics, making learned world models easier for humans to interpret, and making it easier to produce causal explanations for why a robot selected a particular action.
- *Updatability*: This same semantic clarity also makes it easier to update a learned model based on new experience. A local change in the world model might have large repercussions for the policy, possibly changing all of its outputs. Making an update of this kind to an explicit policy would be very computationally expensive. In addition, it is generally efficient to dynamically adapt a declarative model by adding constraints governing the world or the robot.

In general, we aim to get the "best of both worlds," by ensuring that our robots have the generality of a model-based rational action-selection method, but address the common cases in their environment using a combination of

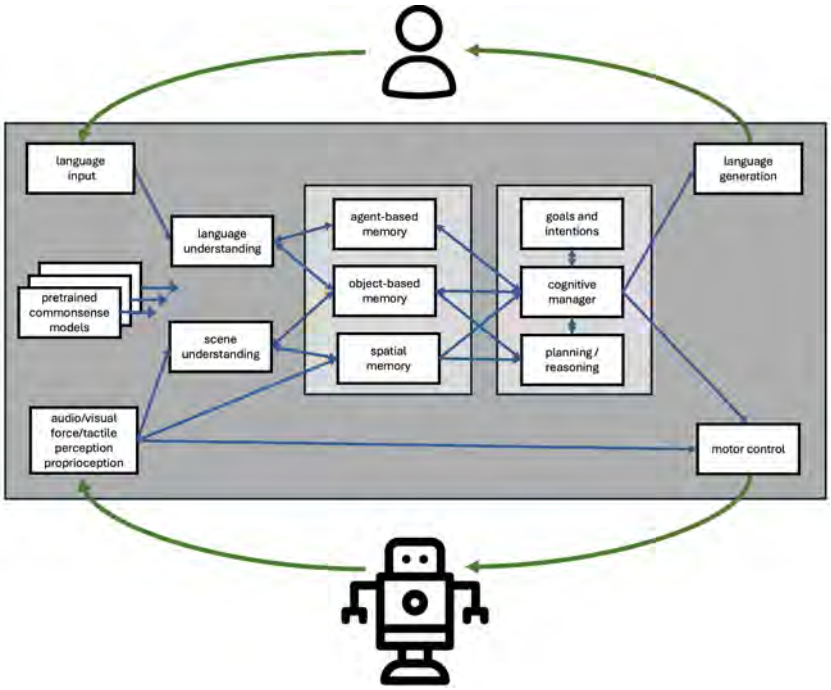

Figure 1.1
High-level modular decomposition of a rational robot policy

fast feed-forward partial policies. These policies can be obtain by online training of neural networks, using solutions found via deliberation as data. The aggregate system would use some form of out-of-distribution detection for the policies, so that when the current situation is not familiar, the slower planning and inference mechanisms are invoked. This process is reminiscent of *routinization* in the cognitive science literature (Anderson 1983) or possibly moving capabilities from Kahneman's "system 2" to "system 1" (Kahneman 2011).

Furthermore, we exploit more subtle forms of learning to make reasoning more efficient, through a meta-reasoning process that enables the system to make decompositions, abstractions, and approximations that focus the inference process on a sequence of tractable sub-problems, as outlined in Section 1.6.

1.3.2 Modularity

Figure 1.1 contains a high-level block diagram illustrating the type of cognitive architecture we believe our robots should have. The overall modularity and the "types" of representations that flow between the modules are the most important aspects, and that there will be a lot of latitude in implementing each individual module. The outermost inputs are low-level percepts such as force, touch and joint proprioception; audio and visual observations, and text and language input. The outputs are ultimately language outputs and low-level robot motor-control commands.

It has become clear that two natural and useful types of representation are natural-language sequences and images or video sequences. These form the inputs and outputs of many highly competent pre-trained models, as well as the fundamental sensed input of cameras and speech/text input and output. One fundamental question in modern AI is whether any other representations are *necessary*.

We won't explicitly address the question of necessity, but will operate on the strong intuition that, for compactness and learnability, the structure of our internal representations should mirror the structure of the world they are representing. For these reasons, we are committed to two additional representational structures:

- A notion of *physical objects* as coherent clumps of matter that tend to stay connected. There is not a single *correct* object-based interpretation (do we speak of individual grapes or the whole bunch?), but we can select a level of abstraction appropriate to our current objectives (Section 1.6).
- Explicitly *three-dimensional* representations of space and shape, including potentially meshes, point-clouds, or grids, indicating the "internal" shapes of spaces the robot can move within and the "external" shapes of objects it can manipulate.

1.3.2.1 Modules There are many ways to choose the particular set of modules in a rational robot architecture; Figure 1.1 contains one basic design, but we are not strongly committed to this particular modularity. The outputs of the spatial, object-based, and agent-based memory constitute the robot's *belief* about the state of the external world, and will necessarily be incomplete and require some explicit representation of uncertainty.

- *Scene understanding*: process input image sequences and produce hypothesized physical objects, including beliefs about shape, material, mass, etc., as well as detections of humans.

- *Language understanding*: process input text and speech and produce information about the beliefs, desired, and intentions of humans in the environment, as well as information about the physical world state.
- *Spatial memory*: construct local high-accuracy maps of free space regions on the scale of rooms; maintain longer-term, hierarchical, more qualitative representation of graphs of regions.
- *Object-based memory*: aggregate output of scene understanding over time, to fuse observations of the same object, reason about how world state changes over time, cache results of expensive perceptual computations.
- *Agent-based memory*: model the other agents in the robot's domain based on linguistic input and inferences about their behavior; information about what the users want the robot to do will be aggregated with "background" utility instilled in the factory to produce objectives for the robot.
- *Cognitive manager*: given the current belief and overall objectives of the robot, decide what action to take next; this is done by framing tractable decision sub-problems and calling the planning module to solve them, and dispatching commands to the motor control (or linguistic output) modules.
- *Planning / reasoning*: find an action sequence (at requested level of abstraction) based on current belief and local goal provided by cognitive manager.
- *Goals and intentions*: based on what we know about the intentions and desires of the humans in the world, as well as built-in value functions and norms, store pending objectives, existing abstract or partial plans, etc.
- *Motor control*: a collection of low-level motor control policies, ranging from classical position or compliant controllers to learned locomotion or dexterous manipulation controllers.
- *Language generation*: produce speech acts to explicitly communicate information or induce action in other agents.

Each of the modules in this architecture require structural design, some engineering specification, learning in the factory, and the ability to adapt on the job. Our own technical work has focused on the decision-making aspects of such systems, so the rest of this paper we focus in particular on the planning component (in sections 1.4 and 1.5) and the *cognitive manager component* (in Section 1.6). The other components are no less important, and we feel strongly that it is critical to work in the context of complete systems that contain at least basic versions of all these capabilities.

1.3.2.2 Feedback Connections It is difficult to show in the diagram, but feedback is critical at many levels of this picture. For example, the scene understanding module should be at least partly query driven, paying attention

to objects that are of interest to the object-based memory or looking for new objects needed by the cognitive manager. The object-based memory should be more active in tracking the state of objects that are relevant to current subgoals specified by the cognitive manager; similarly for the spatial memory. The results of robot controller execution should be fed into the belief update.

1.3.2.3 The Role of General-purpose Pre-trained Models Modern deep learning methods have enormous strengths, which our strategy can use to great advantage.

Computer-vision models for image segmentation and shape estimation are now sufficiently reliable to support the construction of plausible 3D models of a scene from one or more camera images (Agarwal et al. 2024). These reconstructions are, of necessity, partial and not perfectly correct, but coupled with explicit uncertainty quantification (Fang, Kaelbling, and Lozano-Pérez 2024), they can be combined to form an "object-centric" representation that serves as the basis for robust and flexible robot manipulation systems.

Another important aspect of modern learning is the discovery of latent representations, from raw perception, that serve as compact internal representations of state for downstream processing. Monolithic latent state representations will not be consistent with our representational structure, but we exploit the ability to construct "disentangled" latent representations as part of learning world models. Of particular interest, especially for semantically "indexing" into an image or point cloud, are local features (Radford et al. 2021; Oquab et al. 2024; Shen et al. 2023).

Large language models and vision-language models have several important roles to play in our approach. They can be seen as serving as a much more rich and flexible (and fallible) version of the knowledge bases from classical AI.

- They contain factual information that can be of general use, integrated into the inference process, including answers to queries like "Where might I find tea in a kitchen?" or "Under what conditions is it safe to use a fire extinguisher on a kitchen fire?"
- They can suggest courses of action that will guide the search and metacognition processes. Importantly, a learned world-model can be used to validate, elaborate, repair, or reject suggestions from large pre-trained models, thus using them to improve efficiency without endangering correctness.
- They can suggest abstractions and decompositions of state and action that are embedded in nouns and verbs of natural language.

The rest of this chapter focuses on the algorithm-design and learning processes for the sequential planning module. These have been the focus of our

detailed research, and their central role drives design choices throughout the architecture.

1.4 Long-Horizon Decision-Making

The objective of the sequential planning module is to select the next action for the robot to execute. Actions must be chosen in service of achieving the robot's short and long-term objectives, taking into account its current beliefs about the world state, and subject to constraints on online computation time. [2]

As we have argued, for reasons of sample efficiency in learning and space efficiency in policy representation, in many cases it is more effective to structure action selection around online *planning*. A planning system uses a model of the dynamics of its world to consider alternative courses of action (usually sequences of some simple actions), evaluate them in terms of how well they achieve the agent's objectives, select the one that appears best, and execute the first step of that sequence.

We want to emphasize that planning does not require perfect world models, state estimation, or exhaustive search: our architecture is *closed loop*, in the sense that a planning mechanism is used to justify the choice of the next action for execution, but the observations that the robot makes will be used to update its estimate of the world state and induce replanning if the action did not appear to have its intended effect. Thus, errors in interpreting the current world state (perhaps we were wrong about the shapes of objects, or whether a door was unlocked) and approximations in planning (perhaps we didn't consider some unlikely outcome or find the truly best action sequence) can be mitigated by re-observing and replanning.

The problems we are addressing are computationally very difficult: *optimal* planning for a physically complex (many degree-of-freedom) robot in a physically complex domain with mixed discrete and continuous dynamics, large numbers of objects, and substantial uncertainty both with respect to the outcomes of actions and the actual current state of the world, over a very long time horizon (hours or days) is completely computationally intractable. We mitigate this intractability by planning with respect to abstractions and approximations of the true world dynamics, including: assuming independence among parallel and sequential subprocesses, planning hierarchically (in more abstract spaces over a long horizon and fully concretely in the near term), and considering only likely action outcomes and observations. Taken together, and combined with replanning, these approximations allow us to use planning to maintain

2. The work we have done does not yet include considerations of deliberation time, but that is an important question for future work.

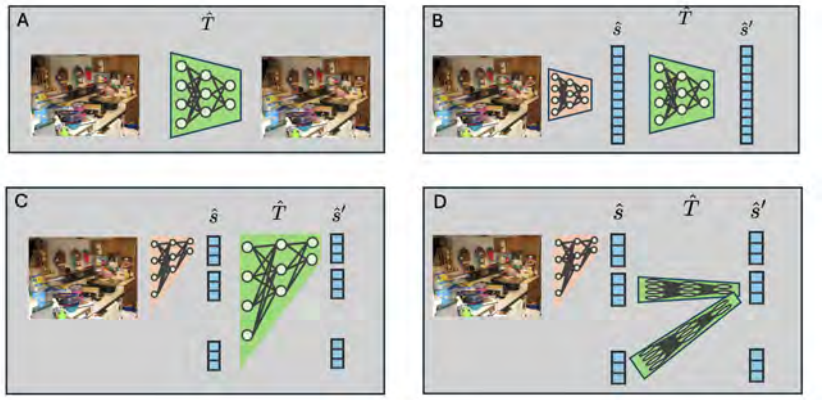

Figure 1.2
World-model representations: (A) monolithic image-to-image, (B) lower-dimensional latent vector representation, (C) factored latent representation, (D) sparse, factored latent representation. Each successive model assumes more about the structure, can be learned more efficiently, and can be used for planning more effectively. Note that the assumption is that underlying structure *exists* in the model, but not what it is, specifically.

goal-direction with respect to very long-horizon objectives, to select actions efficiently in very high-dimensional state spaces, and to be robust with respect to substantial uncertainty in state and outcome.

In the following sections we begin by focusing on the completely observable case, discussing representations and algorithms for efficient planning in huge domains with long horizons. Then we extend our methods to handle uncertainty both in action outcomes and in our estimate of the world state.

1.4.1 World Models with Compositional Structure

A *world model* is a representation of the effects of an agent's actions on the world state for use in planning. Generally it has the form of a mapping from some representation of a world state and an action to a representation of the next world state. A critical question is how to represent such mappings to support computationally efficient planning and data-efficient learning.

Figure 1.2 illustrates a set of representational choices we might make for world models. Most commonly and simply, a raw representation, such as images, can be used directly (Ha and Schmidhuber 2018) for planning. This representational strategy is difficult because it can require a large amount of data to learn and the accuracy of the resulting images tends to degrade upon

repeated application. The video sequences generated by large pre-trained video models such as Sora (OpenAI 2024b) are generally of superficially high quality, but the models are expensive, data-hungry, and still prone to significant errors in actually predicting world dynamics. This strategy is also counter-intuitive: it is hard to imagine planning a cross-country trip, or even going out for coffee, at the level of predicting each specific pixel.

We can mitigate the need for data and make smaller models by finding a lower-dimensional latent representation of the world state, and learn a mapping from one latent representation to the next (Kurutach et al. 2018). This tends to reduce data requirements substantially. Neither of these methods can give any leverage to planning: the only feasible planning strategies are based on forward search, which is completely intractable unless the action space is small and the horizon is very short, or in combination with learned or structure-based search-guidance components.

Factoring

To get significant leverage in both planning and learning, we must expose some structure in the underlying world dynamics. The first step is to *factor* (or "disentangle") the latent representation of the world state into sub-components, and represent the mapping from the entire previous state into each factor of the next state independently, as illustrated in Figure 1.2C. These factors could plausibly be, for example, the indoor and outdoor temperature, number of occupants of the house, etc. This approach may not improve learning efficiency directly, although it does enable the incremental addition of new factors and their predictive models without any interference with previously learned components. Importantly, it already enables an efficiently in planning: the *iterative width* (IW) planners can leverage factored structure to create a domain-independent search heuristic that can substantially speed up planning (Geffner and Lipovetzky 2012).

Further leverage in both learning and planning can be obtained by exploiting structural properties: *sparsity*, in which relatively few factors are changed by any given action and *locality*, in which the new value of a factor depends on the previous values of relatively few factors. Structurally, we might describe the update rule for a single factor by specifying which factors it depends on and specifying some function `f` to compute the new value.

```
if heater_running:
  temperature := f(temperature, window_open)
```

In Section 1.5 we discuss strategies for learning such structures and see that structure can substantially reduce sample complexity. Given a world model of this form, even stronger domain independent search methods can be exploited.

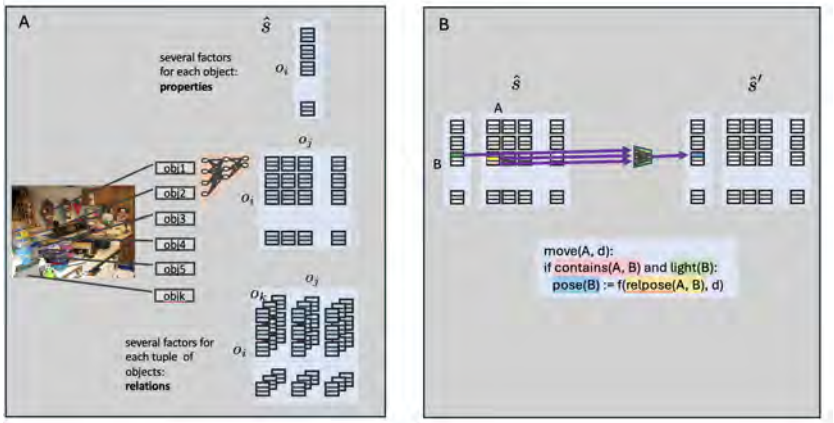

Figure 1.3
A. The lifted state representation uses objects to induce factoring. B. The world model can now be represented using sparse, local *lifted* factors, potentially *instantiated* for any object tuple.

Classical AI planning has explored these methods particularly in the case where the factors have discrete values (Ghallab, Nau, and Traverso 2004), but these methods can be extended to handle continuous-valued factors.

Lifting

In Section 1.3.2 we discussed a representational commitment to physical objects. We can exploit the notion of objects in general (physical objects, to start with, but more abstract objects as well) to extend our approach to factoring. So far, the factors in our representation have referred to state variables that described aspects of the entire world state, such as the indoor temperature. More generally, we can let the objects in our domain induce a factoring, by constructing factors representing *properties* of and *relations* among objects, as illustrated in Figure 1.3A. These properties and relations can be discrete (e.g., A is contained in B) or continuous and multi-dimensional (e.g., the relative pose of A and B).

A lifted representation exposes even more structure and opportunities for learning and planning efficiency. In general, when describing a world model, the particular identities of the objects are irrelevant, so that our descriptions are effectively *quantified* over the objects they mention. For example, we might say: for all objects A and B, if A contains B and B is lightweight, then if I move A, then B will move too. This rule characterizes the behavior of all objects

satisfying the "if" clause. The detailed aspects of the rule can be *learned once* and it can be applied generically in a wide variety of situations, as illustrated in Figure 1.3B. This structure is analogous to that of a graph neural network, in which we learn a small "kernel" describing how to update the state of one node as a function of the values of its neighbors, and then apply it over all nodes in a domain, and use it in domains of varying sizes. We might describe this, formally, as:

```
let B = object
assume light(B)
achieve contains(A, B)
do move(A, distance)
effect: pose(B) = newpose(relpose(A, B), distance)
```

The lifted, factored representation of world models can be seen as a type of "neuro-symbolic" representation: the "symbols" are the object indices and the factors and the "neural" aspect consists of the functions defining the tests (`contains` and `light`) and update functions (`newpose`) in terms of more raw sensory input (or, in our case, state information stored in the object-based memory). In the case that the properties and relations are all discrete-valued, it corresponds to classical AI planning representations such as PDDL (Ghallab, Nau, and Traverso 2004).

We call this generalized form of lifted, factored neuro-symbolic world model a collection of *causal action models.* They enable efficient planning and learning and can be applied to a wide variety of problems (Mao et al. 2024; Silver et al. 2022). The conditions in the `achieve` and `assume` statements are *preconditions* which, if true in some state, guarantee that after executing the action in the `do` statement, the post-conditions in the *effect* statement will be true.

Note that, for the purposes of explaining this type of representation, we are using familiar names for predicates and representing numeric quantities in a standard space. However, we will be able to learn rules of *exactly this form*, in which the learning algorithm has selected its own latent spaces for continuous quantities and its own predicates for providing a factored characterization of object state.

1.4.2 Planning Algorithms

Given factored, lifted world-model representations and a characterization of an intended result, we can generate *plans*, in the form of sequences of calls to primitive controllers, that will move the world from its current state to one satisfying the goal. In the following, we are still making a number of simplifying assumptions, in particular, that the utility of our actions is not time-dependent, that the world state does not change except due to actions of the robot, and

that there are no other agents, except those that can be modeled as a part of the environment. These are all assumptions we would hope to lift in the future.

A subclass of hybrid (discrete/continuous) planning problems that is particularly important for robotics and has been relatively widely addressed is *task and motion planning* (TAMP) and in fact most TAMP algorithms can be applied to the more general class of hybrid planning problems (Garrett, Lozano-Pérez, and Kaelbling 2018; Garrett et al. 2021).

We assume the existence of a set of low-level parameterized "skills," which are closed-loop sensorimotor policies (controllers) that will run for some time and then terminate, generally with some indication of success or failure. An example skill might be to grasp an object in a hand, parameterized by the desired coordinate transform between object and hand. Fundamentally, a plan is a sequence of calls to skill controllers, including values for their continuous parameters.[3]

Most TAMP algorithms combine some form of discrete search for a plausible sequence of skills with some form of continuous constraint-satisfaction problem (CSP) solver for selecting continuous parameters. For example, a plan to place object `A` in container `B` might have the following *skeleton* (sequence of skill invocations):

```
move(init_conf, pick_near_conf, path1)
pick(A, grasp, pick_path)
move(pick_conf, place_near_conf, path2)
place(A, grasp, place_path)
```

and the accompanying CSP

```
pose(A) = pose_A_init
pose(B) = pose_B_init
robot_conf = init_conf
connects(path1, init_conf, pick_near_conf)
collision_free(path1)
valid_pick_cmd(pick_path, A, pose_A_init, grasp )
connects(path2, pick_conf, place_near_conf)
collision_free_while_holding(A, grasp, path2)
valid_place_cmd(place_path, A, pose_A_final, grasp )
is_contained(A, pose_A_final, B, pose_B_init)
```

Each of these constraints is a binary-valued test on high-dimensional continuous variables. For example,

`valid_pick_cmd(pick_path, A, pose_A_init, grasp )`

is true when the robot configuration at the end of `pick_path` corresponds to the specified `grasp` on object `A` when it is placed at `pose_A_init`, which is its current pose.

3. It is not at all obvious where planning stops and "skills" begin. We re-open this question in Section 1.5.

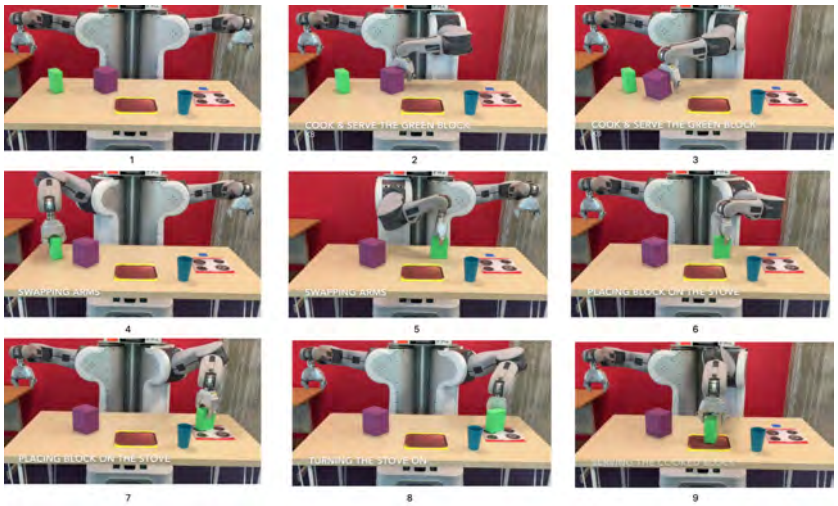

Figure 1.4
A bimanual robot performs task and motion planning and execution. In this example, the objective is to place the green block (which starts at the left of the table) on the "stove" sticker (at the right end of the table), turn on the stove and then serve the block on the central sticker. The robot's left arm cannot reach the right end of the table and the right arm cannot reach the right. The robot must hand over the block from left arm to right arm by first placing it near the middle of the table (frames 4, 5. 6). The purple block would cause a collision with the robot's arm at the chosen hand-off location, so it is first pushed out of the way (frames 2,3); it must be pushed since it is too wide to be grasped. After this, the block can be placed on the stove for "cooking" and eventually "served" (frames 7, 8, 9). (Garrett, Lozano-Pérez, and Kaelbling 2020)

The CSP is typically solved via sampling techniques, in which each constraint type has one or more conditional generative models that, given values for some of the variables in a constraint can produces satisfying values for others. The effectiveness of TAMP methods is highly contingent on the effectiveness of these samplers. These CSP problems can also be approached via optimization-based methods (Toussaint 2015).

Although TAMP problems are highly computationally complex, they can be effectively solved in many practical cases, by exploiting classical search heuristics and well-tuned samplers. Importantly, as well as being able to solve basic "pick and place" problems (Figure 1.4), these methods are applicable to a wide variety of richer problems including planning to find objects (Section 1.4.4.2) and planning in terms of achieving torques necessary to compress springs or lift heavy objects or open sticky mechanisms (Figure 1.5).

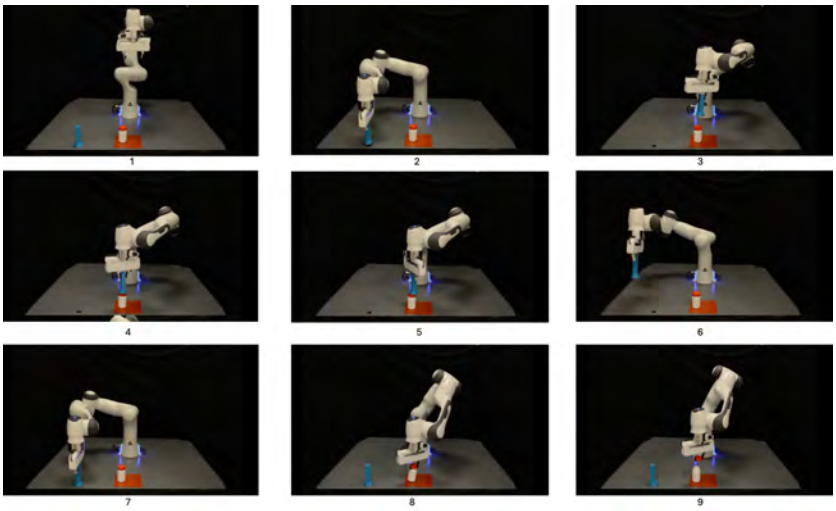

Figure 1.5
Task and motion planning applied to the problem of opening a bottle with a child-proof cap, which must be pressed downward while twisting. The planner has a causal action model for opening the bottle with preconditions described in terms of the necessary forces and torques; it also has has causal action models indicating that it can apply the necessary torque on the cap with its hand or with a tool as long as the bottle is either held in a vice or on the red, high-friction mat. The frames above illustrate one action sequence in which the robot picks up the blue tool (frames 2, 3), presses the tool on the bottle (frame 4), rotates the tool (frames 5, 6), puts the tool away (7), then grasps the cap and lifts it (frames 8, 9). (Holladay, Lozano-Pérez, and Rodriguez 2023)

1.4.3 Temporal Hierarchy

It seems wasteful to plan far into the future if our short-term objective in planning is to determine the next action for execution, and we know that it is likely that unanticipated outcomes will induce replanning. However, if our objective really does require us to select actions now in virtue of the effects they enable days or weeks into the future, what alternative do we have?

One alternative is to make plans at different levels of temporal granularity. In our work on *hierarchical planning in the now (HPN)* (Kaelbling and Lozano-Pérez 2011), we designed and implemented a strategy for using temporal hierarchy for efficiently planning, abstractly, for very long horizons and using those abstract plans to generate more concrete subgoals for more detailed, shorter-horizon plans, culminating in completely detailed plans at a relatively short horizon, the first step of which is executable in the real world.

The key insight is that when subproblems have the *downward refinability* property (Bacchus and Yang 1994) they can be solved independently: given an abstract plan to clean the house, then cook dinner, then serve it, we can plan in more detail for the house-cleaning, largely without concern for the details of the next steps. Of course, we may need to expose some constraints at the high level to enable downward refinability: we should not spend so much time cleaning that we don't have time to cook; we should include a constraint on the cooking not to mess the house up too much, etc. An early principle for constructing hierarchies of this kind comes from Sacerdoti's ABSTRIPS (Sacerdoti 1974): it is fine to postpone the achievement of components of the precondition of an action that can be relatively easily achieved *without changing the state of other components*. For example, it is reasonable to ignore the locations of the serving dishes while cooking, because we know we can move them around later without jeopardizing dinner. Planning in such a hierarchy is illustrated in Figure 1.6.

HPN interleaves the processes of planning and execution as follows: it makes an abstract plan for the top level goal, then picks the first subgoal of that plan, plans in more detail for it, picks the first subgoal of that plan, etc., until arriving at a primitive action for execution. In addition, it retains a "stack" of all the plans it has made in the process, as a kind of summary of the robot's intentions. On the next step, it checks to see if the action had its expected outcome and, if so, executes the next planned low-level step. If we have reached the end of the current plan, HPN pops it off the stack, and makes a more detailed plan for the next step of the plan above. If unanticipated events occur at any point in this process, HPN simply pops plans off the stack until the remaining abstract plan is appropriate in the current situation (e.g., we still plan to make dinner, but give up on the souffle on finding that there are not enough eggs).

This structure enables efficient planning and robust recovery from unanticipated events. It can also enable the robot to exploit new opportunities without an inordinate amount of time spent replanning (Levihn et al. 2013).

1.4.4 Handling Uncertainty

There is a lot more that we don't know than that we do know! The same will be true of our robots. The key to behaving effectively in the presence of uncertainty is taking it explicitly into account in decision-making. In a general-purpose robot system, uncertainty about the domain ranges from local (what are these objects I'm looking at?) to larger scale (what is in the cupboard or around the corner?) to systemic (how do my motor voltages affect my motions?) We can model this all, effectively, as uncertainty about the state of the world, broadly construed.

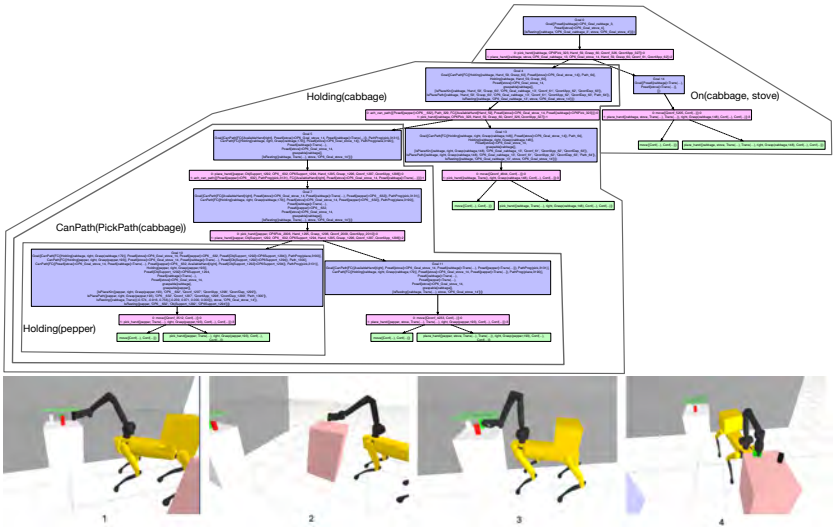

Figure 1.6
Hierarchical planning and execution: the robot's goal is for the "cabbage" (green object) to be on the "stove" (a different table). A combination of fixed obstacles and a movable black obstacle ("pepper") prevent the robot from directly picking up the cabbage. The tree structure indicates the hierarchical reasoning process: the blue nodes represent subgoals and the green nodes are primitive action executions. At the highest level, it plans to pick up the cabbage, then move, and place it on the stove. Then, it plans to achieve the `Holding(cabbage)` condition in more detail. A subgoal of that is to clear a path for picking up the cabbage, which itself requires picking and placing the pepper out of the way. (Kaelbling and Lozano-Pérez 2011)

We divide our thinking about uncertainty into uncertainty about how the future will unfold and uncertainty about the present state.

1.4.4.1 Future-state Uncertainty The simplest kind of uncertainty to model is randomness in action outcomes, under the assumption that the state is always observable; this is the classic Markov decision problem (MDP) formulation. It is important that we continue to express and exploit the structure in the domain, extending the factoring and sparsity ideas to the case of probabilistic outcomes. We can formalize this world model similarly to the deterministic one, except with a probability distribution over the resulting value of each feature that changes.

Making these structural sparsity assumptions, as much as possible while retaining reasonably good predictive accuracy and planning efficacy, will retain

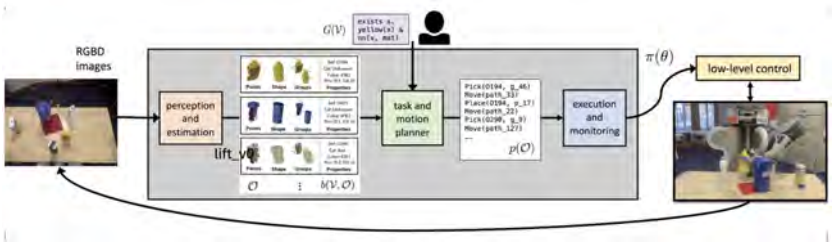

Figure 1.7
The M0M architecture has the form of a closed-loop policy. Given a single input RGBD image, the perception module performs object instance segmentation and shape completion to produce an object-based estimate of the 3D world state. Given a goal in first-order logic, the TAMP system makes a plan under the assumption the perceived world state is accurate. It executes the resulting plan by calling the planned skill controllers, but looks at the scene after each place action, constructing a new world state estimate, and replanning if the result of its actions was not what it intended. (Curtis, Fang, Kaelbling, Lozano-Pérez, and Garrett 2022)

the good sample-complexity results for learning that we have for factored deterministic models (Pasula, Zettlemoyer, and Kaelbling 2007).

Determinization and Replanning

In many domains, especially when there are no "dead ends" (states from which the robot is effectively unable to achieve the goal) or very bad downside risks (of damage to itself or others), it suffices to convert the non-deterministic model to a deterministic one and plan (approximately) as usual. The simplest version of this strategy is to assume that the most likely outcome will occur. It will, of course, be critical to observe the actual outcome and replan if it differs from the intended one. More complex strategies can even include a less-than-most-probable outcome in a plan (e.g., when good luck is the only hope!). The major benefit of determinizing a stochastic domain is that we can use our existing efficient solution methods.

Note that, when using world model rules plus planning as an implementation strategy for a policy, we absolutely rely on our closed-loop hierarchical replanning mechanism to handle stochastic outcomes. This somewhat casual attitude about world models is fine as long as the actions selected usually move the system toward its objective and there is little to no risk of a substantially bad outcome.

We used this strategy to build the *Manipulation with Zero Models (M0M)* TAMP system (Curtis et al. 2022), which supports sophisticated pick and place

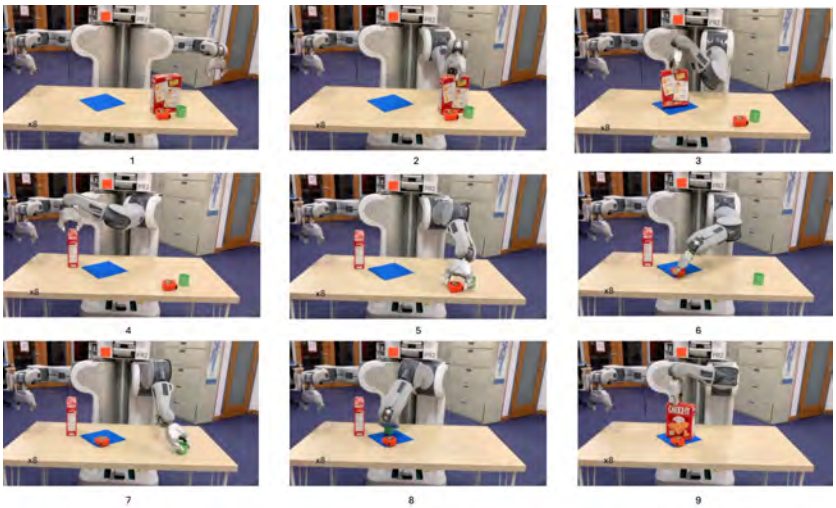

Figure 1.8
The robot is given the goal of putting all the objects on the blue mat. From its perspective, it can only see the large box, and so the estimated state contains a single object, and the TAMP system plans to place it on the mat (frames 1, 2, 3). After doing so, the robot re-observes the scene and discovers two more objects, which had previously been occluded. It constructs a new plan to accommodate all three objects on the mat, which requires first removing the big box (frame 4), because it can't reach over the top to place something behind the box), then placing the two small objects (5, 6, 7, 8), and finally re-placing the large box (9). If the scene were to be perturbed again, the robot would immediately act to re-achieve the goal. (Curtis, Fang, Kaelbling, Lozano-Pérez, and Garrett 2022)

manipulation with no prior 3D models of objects and a rich logic-based specification language for goals. It is structured as a closed-loop policy, as illustrated in Figure 1.7. The replanning loop can cope effectively with a variety of surprises, including some that arise due to partial observability. Figure 1.8 illustrates an example execution history.

Conditional Planning

However, even with a probability-sensitive model, determinization inherently means that the robot is not considering how bad the outcome might be if something goes wrong. Unfortunately, addressing this problem in generality requires a substantial step up in the complexity of planning, because rather than constructing a single action sequence, we have to construct a policy tree, which branches on possible outcomes and selects actions to take in

every case. The complexity of this process can be reduced by using sampling-based approaches, such as sampling-based (Kearns, Mansour, and Ng 1999) or Monte-Carlo tree search (Coulom 2006), but it is hard combine these methods with sparse, lifted representations to plan in very large domains at long horizons, without very substantial computational cost. This is an important area for future work.

1.4.4.2 Partial Observability The move to handling current-state uncertainty is much more substantial. Although in some very simple cases (such as the one illustrated in Figure 1.8), it may suffice to make a single best estimate of the world state and act as if it were true, generally, it is critical to be able to plan to take actions to gain information that will aid in future decision-making to perform well in the presence of current-state uncertainty. Fundamentally, the robot must model the space of its own beliefs about the world state, and plan in that *belief space.* General planning in belief space is wildly intractable, so we seek approximations.

We begin by observing that the *belief-state process*, that is, the stochastic process governing the evolution of the robot's belief state over time, as a function of its actions and observations, can be understood as a Markov decision process in the (generally continuous) space of belief states. The robot starts in an initial belief state b_0, selects an action a_0, and then the world draws an observation from the distribution $P(o \mid b_0, a_0)$. Given this new observation, o_1, we can use Bayes rule to update the belief, obtaining $b_1 = bayesUpdate(b_0, a_0, o_1)$. Armed with this view, we can apply solution methods discussed above for MDPs, including replanning and the most-likely outcome determinization.

Perhaps surprisingly, if operating in belief space (where a goal might now be that the robot's belief is high that it is within some distance d of some target location), then replanning in belief space with determinization can yield effective policies that combine information-gathering and action. Early work by Platt et al. (2010) illustrates the effectiveness of this approach in a simple domain, as shown in Figure 1.9. The driving intuition here is that if the robot *does not* take any sensing actions, it cannot reach its objective of having low uncertainty about its location. Thus, it has to select observation actions, even under the assumption that those observation actions will yield their most likely outcome. If the robot is mistaken, then the sensing action will cause it to update its belief, and then to make a new plan that is more apt given this new information about the world (Figure 1.9). In simple circumstances, this approach has been shown to be provably convergent, and in many other cases it is practically valuable (Platt et al. 2010).

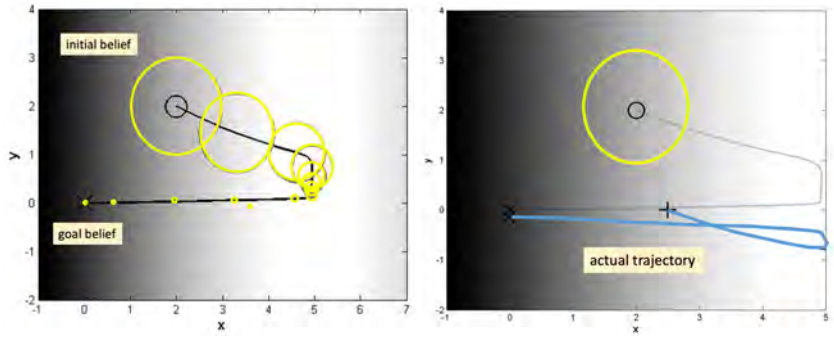

Figure 1.9
The robot in this simple example is a point in the plane. It can sense its location, but can only do so reliable in the "lighter" parts of the domain. Its belief state is modeled as an isotropic Gaussian; the large yellow circle indicates one standard deviation of its initial belief about its location. The goal is to reach location $(0, 0)$ with high certainty. The left panel illustrates the initial plan, made under the assumption that the robot will always make the most likely observation. Under this plan, the robot moves to where it is bright, "hangs out" until its uncertainty is substantially reduced, and then moves toward the goal. The blue path in the right pane illustrates the *actual* robot trajectory that results from executing a replanning strategy in this model. The robot's initial state is, in fact, located at the black plus sign (which is very highly unlikely in the initial belief). The robot begins executing the plan to move right and downward, stopping periodically to re-estimate and replan. It slowly realizes where it is and that is has actually moved farther down than necessary, but recovers and reaches the target. (Platt, Tedrake, Kaelbling, and Lozano-Pérez 2010)

We can determinize this process, as in the previous section, and use causal action models to characterize a lifted and factored model of how the robot's *beliefs* change over time. We will generally have causal action models (CAMs) with belief preconditions (e.g., you have to know the location of an object reasonably accurately before trying to pick it up with a precision grasp) as well as CAMs with belief results (e.g., if you get a visual observation that includes most of an object, the result will be that you know its pose fairly accurately.

Figure 1.10 shows a sequence of manipulation and observation operations produced by a belief-space TAMP system (Garrett et al. 2020) implemented in the PDDLStream TAMP framework (Garrett, Lozano-Pérez, and Kaelbling 2020). It uses a form of determinization (Barry, Kaelbling, and Lozano-Pérez 2011) that uses costs to model the utility of adding uncertain actions to a plan. In particular, if it proposes to take an action that has probability p of success and basic execution cost c, then in the planning search, it assigns cost c/p

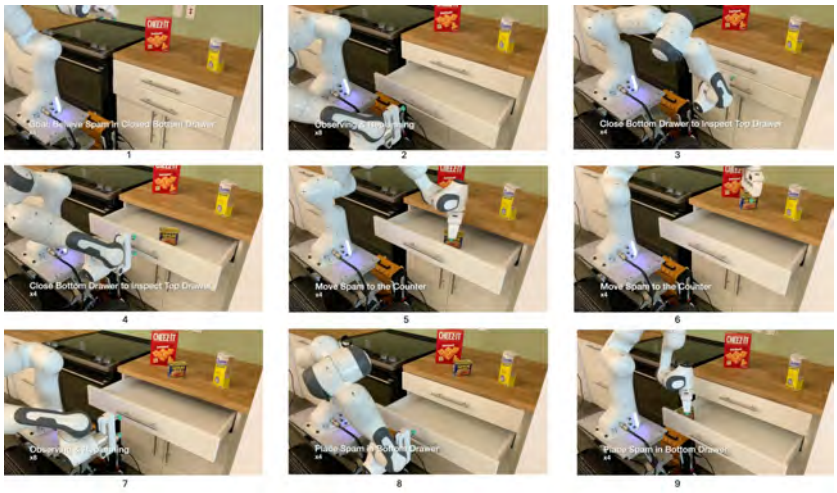

Figure 1.10
An execution sequence of a belief-space TAMP system. The robot's goal is for the can of spam to be in the bottom drawer. The robot knows the shapes of the objects in its domain but not their poses. It initially plans, optimistically, to open the lower drawer and find the spam there (frames 1, 2). Sadly, it is disappointed when it looks in the drawer, and must replan. It decides to look in the upper drawer, which requires reasoning about reachability, and closing the lower drawer first (frame 3). It opens the upper drawer (frame 4) and finds the spam! It then places it on the counter (frames 5, 6), closes the upper drawer (frame 7), opens the lower drawer (frame 8) and places the Spam where it goes (frame 9). Throughout the planning process, the system maintained a "particle filter" estimate of its current belief about the location of the Spam and used it to reason about the effects of its actions and observations during planning. (Garrett, Paxton, Lozano-Pérez, Kaelbling, and Fox 2020)

which is the expected cost of successfully achieving the desired effect under the (unrealistic) assumptions that if the action does not succeed, the world state will remain the same and that successive attempts of the action have independent success probabilities.

Hierarchical replanning is particularly important when planning in belief space, because in many cases there is simply not enough information to complete a detailed plan. Figure 1.11 illustrates a similar process, this time with the robot seeking to place a full oil bottle in a desired location. The robot initially doesn't know what objects exist in the world, nor what their color, type, or mass properties are. It makes plans to look for suitable objects and to weigh them if necessary, ultimately finding a suitable oil bottle and placing it on the

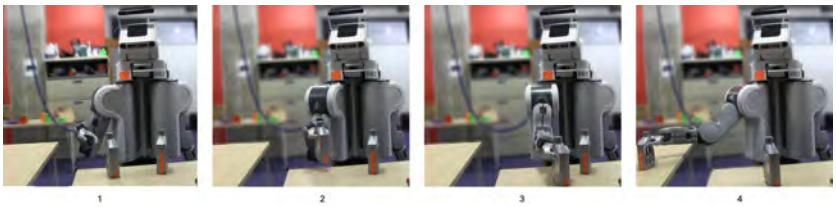

Figure 1.11
Actions in response to the goal of having a heavy oil bottle on the table to the right. The robot picks up one bottle and feels that it is light (frames 1, 2), so it puts it down (frame 3), and picks up the other one and places it on the correct table (frame 4). This process requires planning to sense, belief update, and re-planning. (Kaelbling, LaGrassa, and Lozano-Pérez 2021)

desk. Importantly, HPN postpones detailed manipulation planning until it has observed appropriate candidate objects.

The work described in this section demonstrates that there are computationally feasible method for generating robust and flexible behavior, even in domains with substantial partial observability, via closed-loop perceptually-driven replanning policies. Furthermore, there are no general, robust strategies for solving problems of this kind by direct specification of policies of value functions. In the next section, we show how to learn the world models necessary to support rational decision-making.

1.5 Learning World Models for Decision-Making

In the previous section we showed various approaches for describing highly generalized policies in complex domains with uncertainty. At design time, we determine the "language" that the robot will use to represent what it knows about its domain and some algorithms that it can use to turn that implicit representation into a decision about what action to take. Now we study strategies for using machine-learning methods to acquire causal action models, both in the factory and on the job.

One important observation is that the lifted, factored causal action models that describe aspects of the world model, are relatively easy to learn. Their semantics and correctness are entirely local: given observable data about situations in which the relevant action (e.g. place(X, Y)) was executed and the resulting situations, it is not difficult to learn the conditions and effects of declarative rules.

Causal action models are also natural for incremental learning: we can change one rule on the basis of new information without having to change all

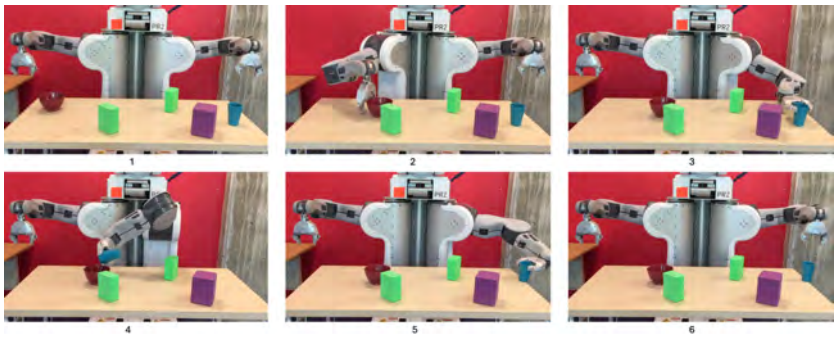

Figure 1.12
An action trajectory resulting from a TAMP system that combines learned causal action models for pushing and pouring with engineered CAMs for pick, move, and place operations. It was given the goal of placing material (which is initially in the blue cup) into the red bowl. Given the initial positions of the cup and bowl, it cannot reach far enough to simply pick up the cup an and pour into the bowl. *With no training on combined operations, but simply reasoning about preconditions and effects*, it plans to push the bowl toward the center of its workspace with one hand and then pour into it with the other. (Z. Wang, Garrett, Kaelbling, and Lozano-Pérez 2021)

the others. We can also add new primitive operations and learn causal action models to describe their behavior, independently of my existing rules. And then, importantly, we can combine these new models with the old ones to solve a whole new class of problems, as illustrated in Figure 1.12.

1.5.1 Learning Causal Action Models

The problem of learning causal action models (CAMs) is at first quite daunting, because there are many components, including

- *Predicates*, which are tests on the belief state, about the relationship among a small set of objects, which may be instantiated as neural networks;
- *State-update functions*, which take as input some aspects of the state of a small set of objects and produce a new value for some aspects of an object, which will generally also be instantiated as neural networks;
- *Controllers*, which are parameterized programs that cause the robot to move; they generally involve high-bandwidth feedback in terms of low-level sensors and may, for example, include compliant control or visual servoing;
- *CAM rule structures*, which provide the causal connection between *preconditions* that, if true in the current belief state, guarantee that after calling the controller with appropriate parameters, the *postconditions* will be true; and

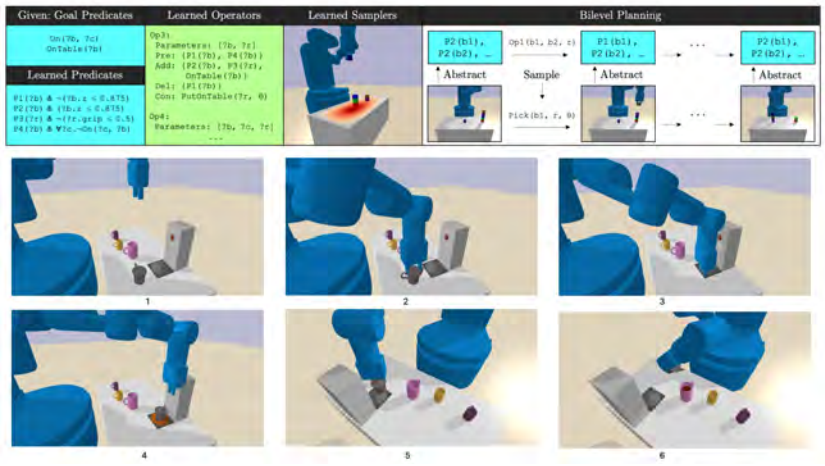

Figure 1.13
An illustration of learning causal action models from experience. Given a small set of predicates that are used to specify goals to the robot (grounded in its perceptual input), a set of low-level parameterized skills, and a set of demonstration trajectories labeled with the skills that were used to produce them and the goal they achieve, we use a combination of program-synthesis and statistical learning methods to synthesize predicates, CAM rule descriptions, and generative samplers for selecting continuous parameters. These learned models can be used to plan for problem domains that are substantially different and more complex than those used for training. (Silver, Chitnis, Kumar, McClinton, Lozano-Pérez, Kaelbling, and Tenenbaum 2023)

- *Constraints*, which are additional relations on parameters in the preconditions, controller, and postconditions that must be satisfied for the effects to be guaranteed.

There are several existing approaches to learning CAMs, each of which assumes some parts are given, and learns the rest.

One frequently pursued approach is to assume that there is a fixed set of controllers, sometimes called *behaviors* or *skills*, which nucleates learning of the other aspects. Early work along these lines includes strategies for learning rule structures, including new predicates, with probability distributions over the outcomes (Pasula, Zettlemoyer, and Kaelbling 2007; Lang, Toussaint, and Kersting 2012). Konidaris, Kaelbling, and Lozano-Pérez (2018) pioneered the "skills to symbols" approach for learning a complete discrete symbolic model of a given set of skills that is suitable for planning under the assumption of complete downward refinability.

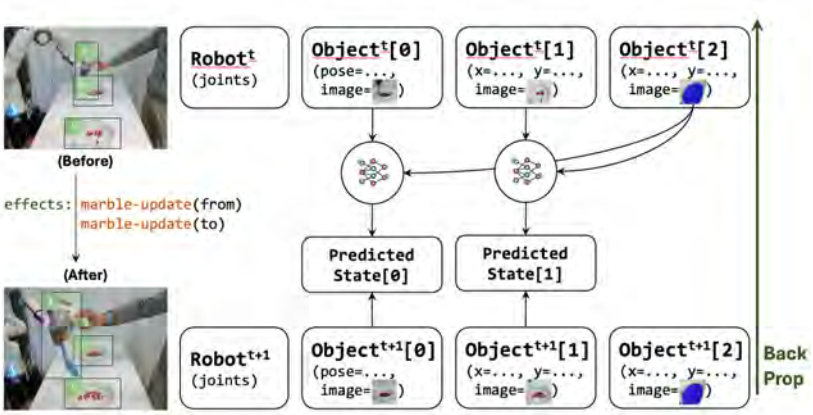

Figure 1.14
Illustration of jointly learning state-update functions and vision-based object-property detectors via backpropagation through a computation graph derived from a training example trajectory. The user labels the trajectory as, for example, scooping red beads and then pouring them into a blue bowl. From this, and examples like these, but no more fine-grained labeling, it learns neural network components that are structured so that the resulting CAMs can be used to solve very novel problem instances. (Mao et al. 2022; Y. Wang et al. 2024)

More recently, work on learning *neuro-symbolic relational transition* (NSRT) rules for bi-level planning (Silver et al. 2022; Silver et al. 2023), illustrated in Figure 1.13 also focuses on learning discrete predicates and rules, but uses them for bi-level planning, which means that the high-level rules serve as guidance for planning in an already known low-level, high-accuracy simulator, and so do not have to be perfectly accurate.

Another approach, called *PDSketch* (Mao et al. 2022), connects more directly with real visual perception. It allows a user to specify the basic structure of a set of CAMs, and uses end-to-end learning to acquire neural-network models of predicate groundings and state-update functions (Figure 1.14). Empirical results illustrate that providing a very small amount of structural knowledge makes the learning process substantially more sample efficient.

A complementary strategy is, given predicates that identify important aspects of the world state, to learn motor-control policies or skills for making those predicates true. This can be done via learning from demonstration or

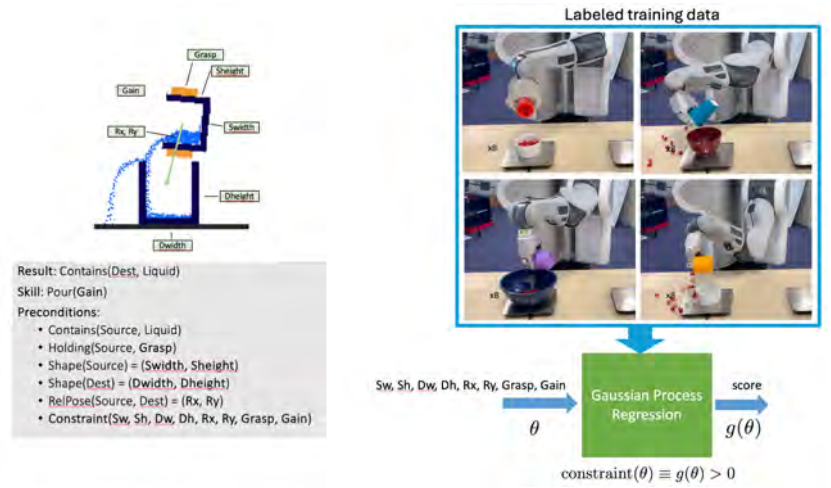

Figure 1.15
Given a new motor-control policy (in this case, for pouring) and a template for a causal action model description (which indicates which properties of which objects are relevant), our objective is to learn a *constraint* function that indicates which combinations of values of parameters describing the initial state, controller, and goal, will result in a successful execution. We use active experiment design, via Gaussian process regression, to select training situations, and use the planner to select object grasps and trajectories during training. This approach can efficiently learn the preconditions and effects of the new skill, enabling it to be immediately combined, via planning, with existing skills, as illustrated in Figure 1.12. (Z. Wang, Garrett, Kaelbling, and Lozano-Pérez 2021)

reinforcement, or simultaneously with predicate and rule learning, given segmented demonstrations (Silver et al. 2022). And, for robotics problems that require constraints on continuous parameters to be considered during planning (for example, where to place the robot base and arm before attempting to invoke a skill for pouring), it is possible to learn a generative model for the constraint while minimizing the online sample complexity (Z. Wang et al. 2021) (see Figure 1.15).

1.5.2 Incremental Learning Cycle

Ultimately, our vision is to combine ideas from all of these approaches, to build a system that incrementally acquires and refines CAMs, starting in the factory and continuing throughout its deployed lifetime. The cycle might operate as follows (Figure 1.16):

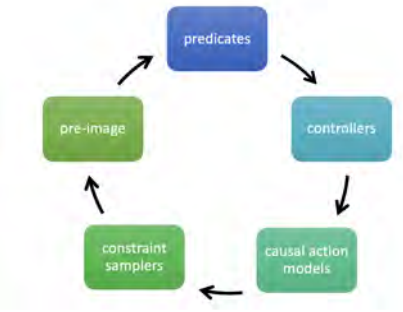

Figure 1.16
Continual learning cycle for causal action models: Innate predicates engender controllers to achieve them; CAMs are instantiated to represent the pre-images of the operations; expressing preimages may require new concepts which are instantiated with new predicates; these predicates may require new controllers, etc.

- We begin with some initial predicates, which might be innate (e.g., whether an object is being grasped or not), or learned via explicit teaching.
- We use a combination of learning from demonstration and reinforcement learning, based on strong control-theoretic primitives, to acquire controllers that can achieve those predicates; these controllers can be relatively short-horizon but one objective is to make set of states from which they can reliably achieve the desired predicate be as large as possible.
- Given a controller and a desired resulting predicate, we can instantiate a partial causal action model.
- Now, we must learn the preimage, that is, conditions under which the learned controller has the desired effect. To the extent possible, these should be articulated in terms of existing predicates on the world state and constraints among numerical parameters in the pre-image conditions, the result, and the controller.
- In some cases, it will be necessary to articulate a novel precondition: e.g., what has to be true of a pitcher for using it to pour causes water to be in my glass? In these cases, we can instantiate a new predicate and learn a test for it, and then go around the cycle again, learning a controller to make that predicate true, etc.

An incremental learning process of this kind is likely to end up with a messy, sub-optimal set of CAMs. We envision a parallel computational process, inspired by the "sleep" phase of DreamCoder (Ellis et al. 2023), that might

notice and clean up repeated structure, find predicates that would be more effective if split into separate components, etc.

An advantage of this overall strategy is its flexibility: engineers could potentially build in predicates, controllers, and CAMs to support basic pick-and-place TAMP operations, robots could learn additional components in the factory, and then a robot fielded in someone's home could quickly and incrementally learn specialized components for cooking their owner's favorite dinner.

1.5.3 Leveraging Pretrained Models

Large vision and language models have an important role to play in the acquisition of CAMs. Our approach is to use these models to guide learning CAMs; however, to retain robustness and reliability, we do not use these models directly to select actions (as is done by (K. Lin et al. 2023)).

The *Ada* method (Wong et al. 2024) uses large language models to incrementally guide learning discrete CAMs as well as their associated controllers. In particular, the LLM is used for translating natural-language goals into formal language for the planner, for suggesting potential high-level task decompositions, and then for converting them into CAM rule structures (including preconditions and effects). This process leverages both the "common sense" causal knowledge contained in the LLM as well as the natural-language concepts that have evolved to be a compact and expressive way of describing important aspects of the world.

Large *vision language models* (VLMs) combine the strengths of LLMs with the ability to interpret and generate images and short video sequences. Given a demonstration of a complex process (such as making tea or assembling a table), a VLM can be used to suggest relevant predicates as well as to (with moderate reliability) serve as the mechanism for grounding those predicates in future applications of planning with CAMs that have been learned based on those predicates (Athalye et al. 2024).

1.5.4 Learning World Models under Partial Observability

In general, learning causal models when the world state is not fully observable is highly computationally complex, requiring probabilistic estimation of the actual underlying state and using expectation-maximization to optimize the world model. This process is intractable for all but the smallest hidden Markov models, and generally plagued by local optima, unless very restrictive assumptions are made.

Humans clearly have substantial partial observability, but also are quite clearly able to learn effective causal models of our world. What is the special property of our world that enables this? One view is that, with effort, we can determine most "everyday" properties of the world we live in: we might not know how much something weighs, or what is inside a drawer, or how hard it is necessary to pull on a door to open it, but with a relatively small amount of local experimentation and observation, we *can* determine these features of the world state to a sufficient degree of accuracy to be useful in planning our actions. The work of Merlin et al. (2024) provides a framework for *locally observable* MDPs and shows that they can be solved efficiently. It seems likely that models such as this can be learned more efficiently, as well.

For these reasons, our view is that it is not generally necessary to explicitly handle partial observability in model learning. For learning in the factory, we may be able to engineer access to privileged information in simulation or run special information-gathering processes in physical environments. For learning during deployment, the considerations relating to explicit information-gathering actions discussed in Section 1.4.4.2 apply.

1.6 Meta-Cognition

So far, we have argued for learning to behave by acquiring a lifted, factored model of how the robot's actions affect the world state or its own belief, and using planning algorithms that operate over approximations of that model to efficiently solve problems. These techniques are helpful, but will not suffice for addressing truly large problems. To do so, we need strategies for improving the efficiency of planning and learning. In the sections below, we discuss two approaches to this problem: improving the speed of solving a particular planning problem, and reducing large problems into collections of smaller ones. We can think of these process as a kind of *meta-reasoning* or *meta-learning* (Russell and Wefald 1991): thinking and learning about how to think. One critical point to keep in mind is that, since metacognition is intended to make the cognitive process more efficient, it is necessary that the metacognitive process itself be highly computationally efficient. This is a good place for using learned (or pre-compiled) policies that run effectively in constant time.

1.6.1 Learning to Select Actions Efficiently

We can use learning to speed up the planning process as well as to "cache" its results. To speed up planning, we retain the causal action model, and still search for a plan satisfying start, goal, and action constraints, but use learned components to guide the search by generating choices (discrete actions, objects

to operate on, continuous parameter values) that are more likely to be useful and by providing heuristic evaluation of intermediate states.

Learning a heuristic or goal-conditioned value-function estimate has been demonstrated to be useful in systems from Alpha Zero (Schrittwieser et al. 2020) to classical planning (Gehring et al. 2022). In TAMP, there have been several systems for learning and using heuristic functions (Mao et al. 2022; Kim and Shimanuki 2019). Such a value function can be used to decide which branches of the search tree to evaluate first when planning.

Another critical part of speeding up search, particularly in continuous domains, is learning *samplers* that select particular objects in the domain (e.g., a container to carry something) or continuous parameter values (e.g., a placement for an object inside a box). Although rejection sampling can generally, eventually, find samples that satisfy a set of constraints, if they exist, it can take a prohibitively long time. We have explored several strategies for learning samplers to improve the efficiency of planning, including doing so while learning preimage constraints (Z. Wang et al. 2021) and adapting samplers online, during the robot's deployment (Mendez, Kaelbling, and Lozano-Pérez 2023).

Sometimes there are sequences or patterns of primitive (or higher-level) steps that together form a "macro-operator" that can be used to achieve a class of goals; rather than rediscover the sequence via planning, we can make a new causal action model that characterizes the preconditions and effects of the entire sequence, and that can be freely combined with other CAMs to plan more complex sequences. We have explored learning sequences, together with specialized samplers for parameters (such as grasps or placements) that make the entire sequence have the desired effect, in the context of tool use for manipulation (Mao et al. 2023; Liu et al. 2024).

Another strategy is to use the results of planning to directly train a policy via supervised learning (Mandlekar et al. 2023). In the setting of an agent that is learning online, we would expect the policy to be *partial* in the sense that it only makes good action predictions in parts of the space for which it has had training data. An approach that we have experimented with, in this case, is to equip the policy with a facility for uncertainty quantification or out-of-distribution detection, and then, in situations where the policy can make a high-confidence prediction, execute the action it suggests; otherwise, run the planner on the world model to get a plan, and both execute the plan and use it as more training data to improve the domain of the policy.

Finally, we can leverage large pre-trained models to provide search guidance at many levels. For example, Yang et al. (2025) used a vision-language model to suggest high-level steps for a very long-horizon plan; these steps served as

subgoals for detailed TAMP planning; the composed system was much more effective than the VLM or the TAMP system alone.

1.6.2 Dynamic Abstraction

One way to approach scaling is to try to make representations and algorithms that will let us directly solve very large planning instances. But we advocate a different way: scaling the huge problem we have before us (managing a household for a month or clearing a disaster area) *down* by breaking it into a sequence of tractable subproblems. There are several potential sources of leverage for doing this. In each case, we make smaller problems.

In our architecture (Figure 1.1), it is the *cognitive manager* module that decides, based on the current goals and beliefs of the agent, whether and how to formulate a planning problem, and ultimately, what actions should be executed via language or motor control.

Temporal abstraction In Section 1.4.3 we discussed a strategy for using hierarchy to make planning efficient. Importantly, though, choices about exactly what hierarchical decomposition to use should be made flexibly, based on aspects of the current state and goal, as well as a sense of the current time pressure. Self-supervised learning, based on previous uses of hierarchical approximations, can be used to train a policy for deciding, for example, how detailed a plan should be obtained before beginning to execute.

Spatial abstraction In general, we would like our robots to operate with horizons of days or weeks, in domains with thousands of objects and a large spatial scale. It is clear that if any aspect of the robot's decision-making process is even *linear* in this spatial complexity, then it will be too slow. However, for any particular planning problem (possibly very detailed and short-horizon; or very abstract and long-horizon) it will be possible to construct an abstract representation of the state space that makes the planning feasible. For example, we might represent and plan in a highly accurate but local model of a room, when determining paths for the robot or placement for furniture, while ignoring the rest of the house or town. But we might change views, and represent the town at the scale of road intersections for the purpose of scheduling errands. Similarly, one set of objects and properties might be critical to a particular subplan, allowing the rest to be ignored. In one pilot project (Silver et al. 2021), we used graph neural networks to predict which objects would be relevant to a given planning problem instance, substantially decreasing running time for the overall planning process.

Teleological abstraction Related to temporal abstraction is a more general notion of *teleological*, or goal-based abstraction. Typical planning formulations have a single conjunctive goal (e.g., the table is set and the dinner is ready); typical Markov decision-process formulations ask the agent to optimize some cumulative reward measure. In real-world long-horizon problems, we require a formulation that is more structured than reward maximization but more nuanced than a single terminal conjunctive goal. More typically, an agent running a household or working to stabilize a disaster area has multiple simultaneous objectives (maintain safety, improve water supply, keep the house clean, etc.) but must be able to reduce that complex global objective into a sequence of much more localized, shorter-term objectives. For example, it might make a decision to focus on fixing a particular water main, which will help with improving the water supply; but it would be necessary to do that work subject to constraints about safety and stability, to prevent making problems worse in other dimensions. This is an important area for future research, in which there is currently little relevant work of our own or of others.

1.7 Conclusions

The key to building robots that have the intelligence, robustness, and flexibility of humans is generality. They must be able to react reasonably to the enormous range of circumstances that may arise in our complex world. It may in principle be possible to achieve such generality via an enormous amount of data, but we advocate for exploring an alternative strategy, in which strong structural priors grounded in math, physics, and cognition provide a basis for extreme generalization from small amounts of data. We argue that modularity, compositionality, and model-based run-time rationality provide the necessary ingredients for the design of robots that can exploit pre-trained models, learn completely novel concepts and behaviors during deployment, and react intelligently to whatever the world may bring.

Bibliography

Agarwal, Aditya, Gaurav Singh, Bipasha Sen, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2024. *SceneComplete: Open-World 3D Scene Completion in Complex Real World Environments for Robot Manipulation.* arXiv: 2410.23643.

Anderson, John R. 1983. *The architecture of cognition.* Harvard University Press.

Athalye, Ashay, Nishanth Kumar, Tom Silver, Yichao Liang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2024. "Predicate Invention from Pixels via Pretrained Vision-Language Models." In *Workshop on Planning in the Era of LLMs (LM4Plan) @ AAAI 2025.*

Bacchus, Fahiem, and Qiang Yang. 1994. "Downward Refinement and the Efficiency of Hierarchical Problem Solving." *Artificial Intelligence* 71:43–100.

Barry, Jennifer L., Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2011. "DetH*: approximate Hierarchical Solution of Large Markov Decision Processes." In *International Joint Conference on Artificial Intelligence (IJCAI).*

Brown, Noam, and Tuomas Sandholm. 2019. "Superhuman AI for multiplayer poker." *Science* 365 (6456): 885–890.

Coulom, Rémi. 2006. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." In *Computers and Games.*

Curtis, Aidan, Xiaolin Fang, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Caelan Reed Garrett. 2022. "Long-Horizon Manipulation of Unknown Objects via Task and Motion Planning with Estimated Affordances." In *Proc. of The International Conference in Robotics and Automation (ICRA).*

Ellis, Kevin, Lionel Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. 2023. "DreamCoder: growing generalizable, interpretable knowledge with wake–sleep Bayesian program learning." *Philosophical Transactions of the Royal Society A* 381 (2251).

Fang, Xiaolin, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2024. "Embodied Uncertainty-Aware Object Segmentation." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Garrett, Caelan Reed, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2021. "Integrated Task and Motion Planning." *Annual review of control, robotics, and autonomous systems* 4.

Garrett, Caelan Reed, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2018. "Sampling-based methods for factored task and motion planning." *The International Journal of Robotics Research (IJRR).*

Garrett, Caelan Reed, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2020. "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning." In *International Conference on Automated Planning and Scheduling (ICAPS).*

Garrett, Caelan Reed, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. 2020. "Online Replanning in Belief Space for Partially Observable Task and Motion Problems." In *International Conference on Robotics and Automation (ICRA).*

Geffner, Hector, and Nir Lipovetzky. 2012. "Width and serialization of classical planning problems." In *European Conference on Artificial Intelligence (ECAI).*

Gehring, Clement, Masataro Asai, Rohan Chitnis, Tom Silver, Leslie Pack Kaelbling, Shirin Sohrabi, and Michael Katz. 2022. "Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators." In *International Conference on Automated Planning and Scheduling (ICAPS).*

Ghallab, Malik, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: Theory and Practice.* Morgan Kaufmann.

Ha, David, and Jürgen Schmidhuber. 2018. "Recurrent World Models Facilitate Policy Evolution." In *Advances in Neural Information Processing Systems (NeurIPS).*

Holladay, Rachel, Tomás Lozano-Pérez, and Alberto Rodriguez. 2023. "Robust Planning for Multi-Stage Forceful Manipulation." *The International Journal of Robotics Research (IJRR).*

Kaelbling, Leslie Pack, Alex Licari LaGrassa, and Tomás Lozano-Pérez. 2021. *Specifying and achieving goals in open uncertain robot-manipulation domains.* arXiv: 2112 .11199.

Kaelbling, Leslie Pack, and Tomás Lozano-Pérez. 2011. "Hierarchical Task and Motion Planning in the Now." In *IEEE Conference on Robotics and Automation (ICRA).*

Kahneman, Daniel. 2011. *Thinking, Fast and Slow.* New York: Farrar, Straus / Giroux.

Kearns, Michael, Y. Mansour, and A. Ng. 1999. "A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes." *Machine Learning* 49:193–208.

Kim, Beomjoon, and Luke Shimanuki. 2019. "Learning value functions with relational state representations for guiding task-and-motion planning." *Conference on Robot Learning (CoRL).*

Kirillov, Alexander, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, et al. 2023. "Segment Anything." In *2023 IEEE/CVF International Conference on Computer Vision (ICCV).*

Konidaris, George, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2018. "From skills to symbols: Learning symbolic representations for abstract high-level planning." *Journal of Artificial Intelligence Research (JAIR)* 61:215–289.

Kumar, Nishanth, Tom Silver, Willie McClinton, Linfeng Zhao, Stephen Proulx, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Jennifer Barry. 2024. "Practice Makes Perfect: Planning to Learn Skill Parameter Policies." In *Robotics: Science and Systems (RSS).*

Kurutach, Thanard, Aviv Tamar, Ge Yang, Stuart Russell, and Pieter Abbeel. 2018. *Learning Plannable Representations with Causal InfoGAN.* arXiv: 1807.09341.

Lang, Tobias, Marc Toussaint, and Kristian Kersting. 2012. "Exploration in relational domains for model-based reinforcement learning." *Journal of Machine Learning Research (JMLR)* 13 (1): 3725–3768.

Levihn, Martin, Leslie Pack Kaelbling, Tomás Lozano-Pérez, and Mike Stilman. 2013. "Foresight and Reconsideration in Hierarchical Planning and Execution." In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Lin, Chu-Cheng, Aaron Jaech, Xin Li, Matthew R. Gormley, and Jason Eisner. 2021. "Limitations of Autoregressive Models and Their Alternatives." In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* Association for Computational Linguistics.

Lin, Kevin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. 2023. "Text2Motion: From natural language instructions to feasible plans." *Autonomous Robots* (November).

Liu, Yuyao, Jiayuan Mao, Joshua Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2024. "One-Shot Manipulation Strategy Learning by Making Contact Analogies." In *CoRL Workshop on Learning Effective Abstractions for Planning.*

Mandlekar, Ajay, Caelan Reed Garrett, Danfei Xu, and Dieter Fox. 2023. "Human-in-the-Loop Task and Motion Planning for Imitation Learning." In *Conference on Robot Learning (CoRL).*

Mao, Jiayuan, Tomás Lozano-Pérez, Joshua B. Tenenbaum, and Leslie Pack Kaelbling. 2022. "PDSketch: Integrated Domain Programming, Learning, and Planning." In *Advances in Neural Information and Processing Systems (NeurIPS).*

Mao, Jiayuan, Tomás Lozano-Pérez, Joshua B. Tenenbaum, and Leslie Pack Kaelbling. 2023. "Learning Resuable Manipulation Strategies." In *Conference on Robot Learning (CoRL).*

Mao, Jiayuan, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2024. "Hybrid Declarative-Imperative Representations for Hybrid Discrete-Continuous Decision-Making." In *Workshop on the Algorithmic Foundations of Robotics (WAFR).*

Mendez, Jorge A., Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2023. "Embodied Lifelong Learning for Task and Motion Planning." In *Conference on Robot Learning (CoRL).*

Merlin, Max, Shane Parr, Neev Parikh, Sergio Orozco, Vedant Gupta, Eric Rosen, and George Dimitri Konidaris. 2024. "Robot Task Planning Under Local Observability." In *IEEE International Conference on Robotics and Automation (ICRA).*

OpenAI. 2024a. *GPT-4 Technical Report.* arXiv: 2303.08774.

OpenAI. 2024b. *Sora: Creating video from text.* https://openai.com/sora.

Oquab, Maxime, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, et al. 2024. "DINOv2: Learning Robust Visual Features without Supervision." *Transactions on Machine Learning Research.* ISSN: 2835-8856.

Partee, Barbara H. 1984. "Compositionality." In *Varieties of Formal Semantics: Proceedings of the Fourth Amsterdam Colloquium,* edited by Fred Landman and Frank Veltman authou. Foris.

Pasula, Hanna M, Luke S Zettlemoyer, and Leslie Pack Kaelbling. 2007. "Learning symbolic models of stochastic domains." *Journal of Artificial Intelligence Research (JAIR)* 29:309–352.

Platt, Robert, Russell Tedrake, Leslie Kaelbling, and Tomás Lozano-Pérez. 2010. "Belief space planning assuming maximum likelihood observations." In *Robotics: Science and Systems (RSS).*

Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, et al. 2021. "Learning Transferable Visual Models From Natural Language Supervision." In *International Conference on Machine Learning (ICML).*

Russell, Stuart, and Eric Wefald. 1991. "Principles of metareasoning." *Artificial intelligence* 49 (1-3): 361–395.

Sacerdoti, Earl D. 1974. "Planning in a hierarchy of abstraction spaces." *Artificial intelligence* 5 (2): 115–135.

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, et al. 2020. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model." *Nature* 588 (7839): 604–609.

Shah, Dhruv, Ajay Sridhar, Nitish Dashora, Kyle Stachowicz, Kevin Black, Noriaki Hirose, and Sergey Levine. 2023. "ViNT: A Foundation Model for Visual Navigation." In *Conference on Robot Learning (CoRL).*

Shen, William, Ge Yang, Alan Yu, Jansen Wong, Leslie Pack Kaelbling, and Phillip Isola. 2023. "Distilled Feature Fields Enable Few-Shot Language-Guided Manipulation." In *Conference on Robot Learning (CoRL).*

Silver, Tom, Ashay Athalye, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2022. "Learning Neuro-Symbolic Skills for Bilevel Planning." In *Conference on Robot Learning (CoRL).*

Silver, Tom, Rohan Chitnis, Aidan Curtis, Joshua Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2021. "Planning with Learned Object Importance in Large Problem Instances using Graph Neural Networks." In *AAAI Conference on Artificial Intelligence (AAAI).*

Silver, Tom, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Joshua Tenenbaum. 2023. "Predicate Invention for Bilevel Planning." In *AAAI Conference on Artificial Intelligence (AAAI).*

Spelke, Elizabeth S., and Katherine D. Kinzler. 2007. "Core knowledge." *Developmental Science* 10 (1): 89–96.

Sutton, Richard S. 2019. *The Bitter Lesson.* http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Toussaint, Marc. 2015. "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning." In *International Joint Conference on Artificial Intelligence (IJCAI).*

Wang, Yanwei, Tsun-Hsuan Wang, Jiayuan Mao, Michael Hagenow, and Julie Shah. 2024. "Grounding Language Plans in Demonstrations Through Counterfactual Perturbations." In *International Conference on Learning Representations (ICLR).*

Wang, Zi, Caelan Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2021. "Learning compositional models of robot skills for task and motion planning." *The International Journal of Robotics Research (IJRR)* 40 (6): 866–894.

Wong, Lio, Jiayuan Mao, Pratyusha Sharma, Zachary S. Siegel, Jiahai Feng, Noa Korneev, Joshua B. Tenenbaum, and Jacob Andreas. 2024. "Learning Adaptive Planning Representations with Natural Language Guidance." In *International Conference on Learning Representations (ICLR).*

Yang, Zhutian, Caelan Garrett, Dieter Fox, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2025. "Guiding Long-Horizon Task and Motion Planning with Vision Language Models." In *ICRA.*