

Deliberation for autonomous robots: A survey

Félix Ingrand, Malik Ghallab

▶ **To cite this version:**

Félix Ingrand, Malik Ghallab. Deliberation for autonomous robots: A survey. Artificial Intelligence, 2017, 247 pp.10-44. 10.1016/j.artint.2014.11.003 . hal-01137921

HAL Id: hal-01137921
https://hal.science/hal-01137921

Submitted on 31 Mar 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Deliberation for Autonomous Robots: A Survey

Félix Ingrand[a,*], Malik Ghallab[a]

[a]*LAAS-CNRS, University of Toulouse, France*

Abstract

Autonomous robots facing a diversity of open environments and performing a variety of tasks and interactions need explicit deliberation in order to fulfill their missions. Deliberation is meant to endow a robotic system with extended, more adaptable and robust functionalities, as well as reduce its deployment cost.

The ambition of this survey is to present a global overview of deliberation functions in robotics and to discuss the state of the art in this area. The following five deliberation functions are identified and analyzed: planning, acting, monitoring, observing, and learning. The paper introduces a global perspective on these deliberation functions and discusses their main characteristics, design choices and constraints. The reviewed contributions are discussed with respect to this perspective. The survey focuses as much as possible on papers with a clear robotics content and with a concern on integrating several deliberation functions.

Keywords: Robotics; deliberation; planning; acting; monitoring; observing; learning.

1. Introduction

We are interested here in autonomous robots in industrial, service or field applications, and in robotized systems such as spacecrafts, AUVs, UAVs or self-driving cars. These systems interact *autonomously* with their environment through sensory-motor capabilities and may have to *act deliberately* in order to fulfill their mission. Acting deliberately means performing actions that are motivated by some intended objectives and that are justifiable by sound reasoning with respect to these objectives. Deliberation is any computational function required to act deliberately. From the perspective of robotics, the interest is not on deliberation *per se* but on acting deliberately.

Robots without autonomy do not need much deliberation. These robots extend the acting capabilities of human operators who are in charge of decision making, possibly with the support of advice and planning tools, e.g., as in surgical robotics and other teleoperated applications. Autonomous robots may or may not require explicit deliberation. Deliberation is not needed for autonomous robots working in fixed, well-modeled environments, as for most fixed manipulators in manufacturing applications. Neither would deliberation be required if all feasible missions consist of repeating a single task, e.g., as for a vacuum cleaning or a lawn-mowing robot. In these and similar cases, deliberation takes place at the design stage. Its results are directly implemented in the robot controller.

Autonomous robots facing a *diversity of environments*, a *variety of tasks* and a *range of interactions* cannot be preprogrammed by foreseeing at the design stage all possible courses of actions they may require. These robots need to perform some explicit deliberation. In short: *autonomy plus diversity entail the need for deliberation*.

Deliberative robots are desirable in applications where the environment is open-ended and largely unknown and teleoperation is constrained, as in most exploration robotics. They are needed in service applications in dynamic and changing environments with semantically rich tasks and human interactions. Domestic and personal robots are demanding in deliberation capabilities. Even for applications in closed and well structured environments with a reduced range of tasks, where engineered robotics operations are feasible and have been demonstrated successfully (e.g., in

*Principal corresponding author.
Email addresses: felix@laas.fr (Félix Ingrand), malik@laas.fr (Malik Ghallab)

automated warehouses), deployment and adaptation costs can be reduced by deliberation capabilities. Deliberation can endow a robotic system with extended, more flexible and robust functionalities; it can reduce its deployment cost.

Deliberation is a very active and critical research field in intelligent robotics. It covers a wide spectrum of problems and techniques at the intersection of Robotics and Artificial Intelligence. Deliberation has to closely interact with sensing and actuating. It does not stop at planning and is not reducible to a single function. It critically requires the coherent integration of several deliberation functions. It involves numerous open problems ranging from knowledge representation, acquisition and verification, to scalable reasoning algorithms and architecture issues.

This research field is of interest to several communities in Robotics and Artificial Intelligence. It is covered by a vast literature. Several books and survey papers review the state of the art for a few focused deliberation functions, e.g., planning [91], monitoring [169], goal reasoning [213], recognition of actions and plans [129], architectures [127], or human–aware navigation [130]. A historical perspective of the links between Robotics and AI, from which this survey paper stems, is sketched in [107]. However, to our knowledge there is no global overview of deliberation functions in robotics, of the challenges they pose and the techniques needed for their design and implementation. The lack of a global vision hinders progress in the field toward coherent and more integrated contributions.

The ambition of this paper is to contribute to such an overview. It undertakes this objective by surveying some of the contributions to deliberation functions and putting them into perspective with respect to a global understanding of deliberate action and its integrative requirements. To keep a reasonable scope, we will not survey human–robot and multi-robot interaction and cooperation issues (but of few references relevant beyond interaction issues). Despite this focus, the covered field is not easily amenable to a unifying perspective. It remains excessively broad, fragmented in topics and approaches, but also in time: often, the same line of research spans and matures over a dozen years and as many publications. We believe that taking a global view to deliberation in robotics is a useful step for understanding the state of the art and permitting further advances.

This paper does not propose a unifying theory of deliberation. It presents a comprehensive view of deliberation functions in robotics. It discusses their main characteristics, design choices and constraints for achieving them. The surveyed literature for each deliberation function is discussed with respect to this global perspective.

Given the breadth of the field, this survey cannot exhaustively cover all relevant contributions. We restricted ourselves to papers on deliberation functions with a clear robotics content, and, whenever possible, with a concern for the integration of several deliberation functions. Some of the reviewed contributions are only briefly mentioned, while representations and techniques for a few typical approaches are informally presented.

The next section presents a global view of deliberation functions and their requirements for various robotics applications; it outlines the main options faced by a designer of a deliberation system. The rest of the survey develops an analysis of several contributions to deliberation functions (Sec. 3 to 7). Two main functions, namely planning and acting, are extensively discussed, while the survey is more illustrative for the monitoring, observing and model acquisition functions. However the corresponding sections are essential to the global view advocated for here. This analysis is followed by a discussion and synthesis section (8), then a conclusion with a perspective on future research.

Beyond the wealth and variety of approaches, the take-home message of this survey can be sketched as follow:

- deliberation in robotics does not reduce to task planning,
- acting is a context and task dependent reaction to events and refinement of planned actions into commands,
- hierarchy of action refinements over several levels of state and action spaces is essential,
- monitoring needs strong links to prediction in planning and acting,
- observing requires significant deliberation, it does not reduce to signal processing,
- integrated and consistent representations and the acquisition of corresponding models are a clear bottleneck.

2. Design of a robotics deliberation system

2.1. Deliberation functions

Let us first clarify our terminology. An *action* is a process that changes the state of the robot and its environment; this includes perception and information gathering actions. An action can be viewed at different levels of abstraction *hierarchy*. It is *primitive* at some level and *compound* at the next level. E.g., open(door) is a primitive action at an abstract level and a compound task to be refined, when needed, into a collection of concrete actions. A *command* is a primitive action at the lowest level of the hierarchy. Commands can be directly executed by the robot platform. The

platform is a collection of devices and sensory-motor functions integrating sensing and closed-loop actuation. Choosing the appropriate command and monitoring that the context remains within a command functioning envelope is part of deliberation. The execution of a command does not need deliberation capabilities. For example, grasp(handle) is a command which, under appropriate conditions, can be directly executed by a robot platform. A *plan* is an organized collection of actions. A *policy* is universal plan, i.e., a mapping from states to actions. A *skill* is also an organized collection of actions. It differs from a plan in the following: its actions are at a lower level; its primitives are commands; its control structure can be richer than that of a plan. A skill is retrieved from a library of skills (hand-programmed or learned) while a plan is synthesized by a planner. A *task* is the specification of a mission or objectives to be achieved by the robot, in possibly different forms and abstractions levels, e.g., as *goal* states. An *event* is an occurrence of a change in the state of the robot and its environment; it can be controlled or exogenous, expected or unexpected.

Most of the literature on deliberate action is focused on planning. However, planning alone is not sufficient for achieving deliberate action. Acting is accomplished by exerting controlled forces on the environment. It is implemented at the lowest level by online processing of streams of signals, from sensors to actuators. These streams achieve feedback loops robust to the discrepancies between the world and its models. Planning, on the other hand, implements a mapping of its input, a predictive model and a problem, into an output plan expressed in some representation. Because of the limitation of predictive models and complexity constraints, planning has to admit several simplifying assumptions. As a result, a synthesized plan provides a projected course of action in some abstract representation. It is useful but rarely sufficient for acting deliberately.

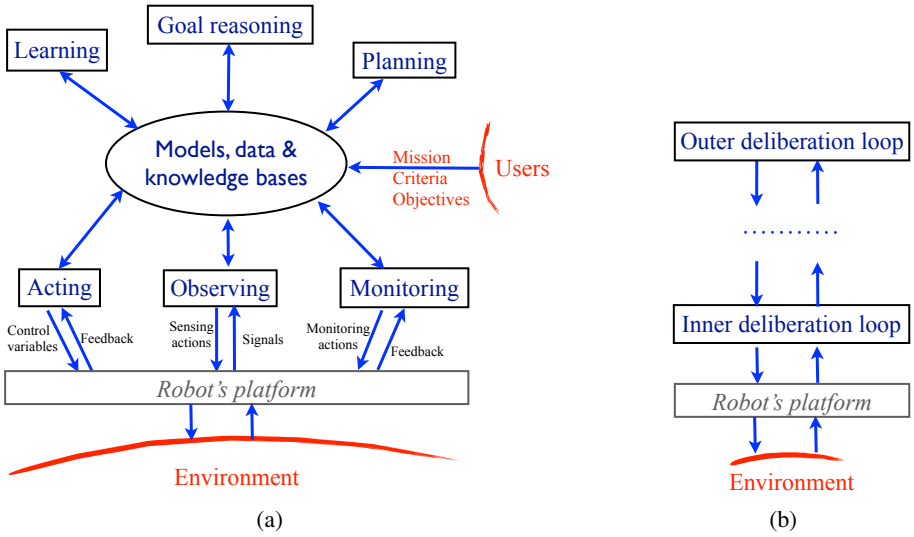


Figure 1: (a) Schematic view of *deliberation functions* (interactions between functions is oversimplified; more complex architectures are briefly discussed in Section 2.3); (b) Nested closed-loops implementing these functions.

There is more to deliberate action than just planning in an abstract space. Several distinct functions can be required for acting deliberately. The borderlines between these functions are blurry and depend on specific representations, implementations, and architectures. However, it is conceptually clarifying to distinguish the following six *deliberation functions* (Fig. 1(a)):

- Planning: combines *prediction* and *search* to synthesize a trajectory in some abstract action space based on predictive models of the environment and feasible actions in order to achieve some purpose.
- Acting: *refines* planned actions into commands appropriate for the current context and *reacts* to events; both refinement and reaction may rely on *skills*, i.e., a collection of closed-loop functions. A skill processes a sequence (or a continuous stream) of stimulus input from sensors and generates output to actuators, in order to trigger motor forces and control the correct achievement of chosen actions.
- Observing: *Detects* and *recognizes* features and relations characterizing the state of the world, as well as events, actions and plans relevant for the task. Observing combines bottom-up processes, from sensors to meaningful data,

with top-down activities such as focus mechanisms, sensing actions and planning for information gathering.

- Monitoring: *Compares* what is *predicted* regarding the robot activities to what is *observed* in the world. It detects and interprets discrepancies, performs diagnosis and triggers recovery actions when needed.
- Goal reasoning is *monitoring at the mission level*. It keeps commitments and goals in perspective; it assesses their relevance from the observed evolutions from new opportunities, constraints or failures; it entails from this assessment whether some commitments should be abandoned, and when and how to update the current goals.
- Learning: allows a robot to *acquire, adapt and improve* through experience the models needed for deliberation. Active learning induces a robot to act deliberately in order to better learn.

These deliberation functions are interfaced with the robot's platform. The frontier between sensory-motor functions and deliberation functions varies widely depending on the complexity of the planned actions and the variability of the environment. For example, in a navigation task, motion control along a path is a sensory-motor function as long as the ground is nominal (i.e., firm, stable). When the robot gets out of the envelope of a nominal control mode, deliberation is needed. Sensory-motor functions should not be perceived in a reductionist view as simple feedback control loops. They often require *feedforward* prediction and monitoring of their functioning conditions (e.g., checking that the control state remains in the envelope), and complex interactions with deliberation functions.This is illustrated in neuroscience models of human and primate motion, e.g., [195]. Furthermore, learning capabilities change the frontier of when and where explicit deliberation is needed. E.g., in a well-known environment, a navigation skill to the same usual destination can be compiled down into a low-level control with pre-cached parameters.

Deliberation is necessarily confined within bounded computational resources. A *metareasoning function* can be needed for trading off deliberation time for action time, given current constraints and objectives at stake. Critical activities require careful deliberation, while less important or more urgent tasks may not need, or allow for, more than fast approximate solutions, at least for a first reaction.

Finally, note that Fig. 1(a) is not complete: *Interaction*, in particular human-robot interaction, is clearly a deliberation function. This is illustrated in the capabilities integrated in several systems, e.g., [193]. In some cases interaction can be used to get human help in deliberation problems, e.g., as in [187]. To keep a reasonable scope, interaction is not covered in this survey.

2.2. Characteristics and deliberation requirements of robotics applications

Deliberation is entailed by autonomy and diversity; but these are not binary properties, either true or false. When deliberation is needed, it may have different characteristics depending on the requirements of the robotics application at hand. Let us look at a few typical classes of applications.



(a) (b) (c)

Figure 2: (a) An intervention robot (LAAS), (b) A rescue UAV (ONERA), (c) A user's assistant robot (LAAS).

An *autonomous satellite* has a fairly stable and predictable environment. It has a well focused set of tasks, e.g., to use its imaging devices efficiently to respond to received queries given expected or perceived cloud coverage, and to manage its communication, processing and energy resources. It can plan and execute stable plans over long horizons. Its deployment is costly, which justifies extensive engineering for an optimized behavior, with diagnosis and self repair.

An *autonomous space exploration* robot, e.g., Curiosity, faces a wider diversity of tasks (navigation, observation, sampling, communication). Its environment is variable but fairly predictable; it has a low semantic complexity. The robot controls its sensory-motor devices according to its own plans. It monitors its activity, adapts and repairs its plan in unforeseen situations. It considers opportunities (e.g., an interesting rock) and manages its mission. All these deliberation functions are integrated on board and properly scheduled with respect to limited computational resources. They share models (perception models, action models, skill models, monitoring models, etc.), which need to be consistent with respect to each others. Here also the mission can afford extensive engineering and development.

A *field robot for surveillance and rescue missions* (Fig. 2(a)) has similar features, but it faces richer and more variable environments, less predictable and less easily modeled. Cooperation and interaction with human and other robots is an important feature in these applications. Similar remarks hold for UAVs (Fig. 2(b)) in rescue missions. Further, a UAV may have to face faster dynamics in its tasks and environments.

A *self-driving car* has also to deal with high dynamics, but for a very focused task in well structured environments. Critical safety issues are a strong constraint on the deliberation functions.

A *service or domestic robot* (Fig. 2(c)) does not have to cope with high dynamics. It is faced with very rich environments and a broad diversity of tasks. Its deliberation functions should allow it to reason about its mission (e.g., assisting the user), to plan its tasks (e.g., to serve a simple meal), to act according to its plan, refining actions into closed loop commands and reacting to events, to perceive its environment for building and maintaining its models. Engineering costs should remain low. Learning is important (e.g., adapting its skills to the particulars of the environment). Finally, man-robot interaction is essential for service and domestic robots.

Manufacturing robots used to be in protected unchanging environments, without much need for deliberation. This is progressively changing. There is a demand for very flexible robots that are easily adapted to a broad range of tasks and seamlessly integrated in any working environment. This is the more true for industrial mobile robots viewed as a worker's assistant in his daily chores. This class of applications is in many aspects similar to service and domestic robots, with more emphasis on cooperation and a more focused set of environment and tasks.

	(i) Environment Variability	*(ii)* Task Diversity	*(iii)* Semantics	*(iv)* Dynamics	*(v)* Partial Observability	*(vi)* Cost & Criticality	*(vii)* Interaction & Cooperation	*(viii)* Level of Autonomy
Manufacturing robots	*l*	*l*	*l*	*h*	*l*	*m*	*l*	*l*
Industrial mobile robots	*m*	*m*	*m*	*m*	*m*	*m*	*h*	*l*
Exploration & rescue robots	*h*	*m*	*l*	*m*	*h*	*h*	*h*	*h*
Service and domestic robots	*h*	*h*	*h*	*l*	*h*	*l*	*h*	*m-h*
Autonomous spacecraft	*l*	*l*	*l*	*l*	*h*	*l*	*l*	*m*
Autonomous aerial vehicles	*m*	*m*	*m*	*h*	*h*	*h*	*m*	*m-h*
Autonomous cars	*m*	*l*	*l*	*h*	*m*	*h*	*m*	*l*

Table 1: Features of a few classes of robotics applications (*l*: low, *m*: medium, *h*: high).

We can analyze the deliberation requirements of a robotics application through eight main features: *(i)* how variable is the environment, *(ii)* how diverse is the set of tasks, *(iii)* how rich are the semantics, *(iv)* how fast are the dynamics of the environments and tasks, *(v)* how partially observable is the environment *(vi)* how high is the cost or criticality of the system, *(vii)* how important are interaction and cooperation with people and/or other robots, and *(viii)* how autonomous is the robot in carrying out its mission, e.g., choosing and committing to goals. These requirements are summarized in Table 1, considering three qualitative levels for each feature in typical applications for each class.

Application features entail constraints and requirements on deliberation functions. For example, monitoring depends on the environment variability; goal reasoning is desirable for generic missions and high degree of autonomy; safety critical applications demand formal methods and tools for the specification and verification of some of the functions to be implemented. The above table is useful for analyzing and characterizing known deliberation systems.

The surveyed systems will be characterized with respect to these features; in particular we'll refer to applications marked "*h*" in one or a few columns. However, most of these systems aim at being generic over a range of application classes. Even if they are motivated by and experimented with in a particular class, they propose ideas and techniques that can be relevant beyond their motivating applications. Furthermore, application features constrain deliberation functions, but they leave room for several design choices, e.g., with respect to knowledge representations used and

how deliberation functions are organized. Let us discuss these options.

2.3. Design options of a deliberation system

The integration of planning and acting functions is a critical issues in the design of a deliberation system. To clarify the range of design options, let us oppose two main views emphasizing respectively planning and acting.

Emphasis on planning. This option assumes full predictive, possibly nondeterministic models of the world, including its dynamics and possible effects of exogenous events. It needs *reversible* models of the sensing capabilities of the robot, i.e., how reality is mapped into percepts, as well as reversible models of its actuating capabilities, i.e., how commands are mapped into forces and motion. Inverse kinematic models map desired configurations into commands achieving them; inverse sensing models map sensed features into world objects. It requires a planner that takes as input these global predictive models together with the specifications of the mission, and provides as output a plan or a policy, mapping perceived states (or *beliefs*) into lower level commands, possibly with some guarantee of achieving the mission or meeting some optimality criteria. Under this view, acting is a simple triggering of commands given by the planned policy: at each step the robot senses, i.e., reads the current observation variables, applying the corresponding commands specified by the policy. Note that a discrete unit time approach has to be further extended for dealing with domains involving rich temporal and real-time constraints.

This principled view inspired the design of a few early deliberation systems. It had the merit of clarifying, at the conceptual level, what was desired and how far from the ideal target the achievements were. However, the applicability of this idealistic view is limited by the low predictability of most environments, the tremendous difficulties of specifying the needed models, and the computational complexity of planning.

Emphasis on acting. Designers addressed the previous issues by the usual decomposition strategy, i.e., decompose the deliberation burden between a planning phase and an acting phase, and rely on restrictive assumptions at one end or the other. In the spectrum of possible decompositions and assumptions, classical planning is the most restrictive. It assumes an *open-loop* robot in a *deterministic*, *static* environment (no exogenous events), which is fully *observable* (for the initial state, the other states being entirely predictable). Consequently, acting has to handle *nondeterministic*, *partially observable* and *dynamic* environment models through *closed-loop control*. This option puts most of the burden of deliberating with the real world constraints (noisy sensors and actuators, imperfect models) on acting.

This view drove the design of several implementations. It remains popular (admittedly in the automated planning community, but much less in the robotics community). It led to the development of scalable classical planning techniques. But its usefulness proved to be limited by the complexity of the acting system, the narrow effectiveness of planning and the weakness of the produced plans.

Balanced decomposition. In summary, the first view relies entirely on planning, the second one depends mostly on acting without much efficient help from planning. A major design option is how to better decompose the complex deliberation problem between planning and acting, how to balance the process of choosing and undertaking actions between different levels of abstraction and granularity, from the mission level to the sensing and command level. At least three levels are needed (including the platform level in charge of executing the commands, outside of the scope of deliberation). In principle, more than three levels can be considered. Most approaches consider just three levels, corresponding to the functions we labeled as planning and acting, in addition to the platform.

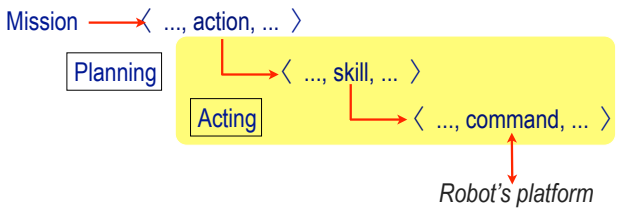


Figure 3: From actions to skills and commands.

Planning synthesizes a trajectory in an abstract action space predicted to achieve (part of) the mission. Acting further refines each abstract action into one or several steps, called *skills*. Skills are plans with possibly concurrent steps whose primitives are *commands* (Fig. 3). Planning requires *descriptive models* giving the requirements and possible effects of actions (*know-what*). Acting requires *operational models*: how to refine actions, which skill to choose and when, how to react to events (*know-how*). This view is further developed in [92].

A balanced decomposition should supply predictive capabilities to planning as well as to acting. Both may perform look-ahead operations, with various planning techniques. It is important to keep in mind that planning and acting may work in distinct state spaces and action spaces. However, they may both use the same representation and possibly a unified model. One may characterize ideas and approaches for specifying deliberation models for planning and acting into the following categories:

- *Single model* approaches rely on a unified domain model merging the *know-what* specification of descriptive models and the *know-how* specification of operational models. Abstraction and hierarchical methods are the usual tools for decomposing the problem. Planning is performed down to some level of abstraction. Abstract actions in plans are further refined into primitive commands. Techniques such as Hierarchical Task Networks, Angelic Hierarchical Planning, or Hierarchical MDPs may support these approaches.
- *Multiple model* approaches use distinct models: an abstract descriptive model for planning and one or several operational models for acting to refine each plan step into skills. These approaches may use either *(i)* a *uniform knowledge representation* for specifying planning and acting problems at different levels of granularity, e.g., formal action languages, or *(ii) heterogeneous knowledge representations* at various levels, e.g., precondition-effect operators for planning, procedures or rules for acting.

The refinement from plan steps into skills involves several other considerations:

- *How close to the platform* are the synthesized plan steps. This property is context dependent: the same action, e.g., moveto, can be a directly executable command in some context and may require further deliberation in another one.
- Are the skills *hand-programmed* vs *automatically synthesized*. The synthesis of skills from the specification of sensory-motor function and/or with machine learning techniques allows for easier engineering design, for verified properties and improved integration between planning and acting.
- *How specific to the current mission and environment* are the skills. Their adaptation may range from the choice of parameters in generic skills, to the online synthesis of skills needed for the task.
- *How flexible* is the generated plan. A partially specified plan leaves freedom for handling some level of uncertainty and nondeterminism and adapting to the execution context.

Architectures. The choice of an appropriate architecture is another major design issue. A robot implements its deliberation online. Hence all approaches have to interleave deliberation functions through some form of *nested closed loops* (Fig. 1(b)): acting is performed at an inner loop level while planning is processed at an outer loop level. Various architectural paradigms have been proposed to address the integration issue (see survey [127]):

- *Hierarchical architectures* are probably the most widely used in robotics [108, 161, 63, 37, 85, 126]. They organize the software into a few layers with different temporal requirements and abstraction levels: a platform layer containing the sensory-motor modules, and one or several layers for deliberation functions. Often, different representations are used in these layers, which makes global consistency checking quite difficult. Their benefit is that they clearly identify and organize the sensory-motor components and the deliberation components. However, these architectures are not easily adapted to functions distributed over several layers (e.g., perception, motion planning, etc.).
- *Reactive architectures* are composed of input/output automata implementing loops from sensors to effectors. These automata can be organized hierarchically and can inhibit or weight each other. Initial instances of reactive architectures, e.g., the subsumption architecture [32], were mostly at the platform level without much concern for deliberation. More recent extensions have added deliberation functions (e.g., [134, 170]).
- *Teleo-reactive architectures* are more recent [156, 74, 147]. They propose an integrated paradigm based on planning–acting components distributed at different levels, using different horizons and time quanta. They use a single representation and provide a continuum between the components (from the high level planner dealing with mission state variables, to the low level reactor modeling the physical state variables). They offer a clear advantage with respect to the consistency of the system, but their scaling up to complex and highly dynamic environment remains an issue.

- *Open architectures* have recently appeared to fulfill the new needs of service and personal robots in open environments with rich semantics. Their distinctive characteristic is to allow a robot to draw data and models over the web, using in particular semantic web techniques. Several example are covered in Section 3.4, e.g., the RoboEarth, [218], KnowRob [208] and CRAM [18] systems, which rely on OWL, the standard Web Ontology Language.

It is important to note that the above integration and architecture design issues are relevant for all deliberation functions. Monitoring and observing, performed at an inner loop level, can benefit from automated synthesis and adaptation to the context. Goal reasoning and learning, at the outer levels, are heavily conditioned by the representations used in other functions. These issues will be further discussed in the remainder of the paper.

2.4. Global view of surveyed systems

Systems	**Associated components**	**Deliberation Functions**	**Knowledge Representation**	**Robotic Domains**	**Architecture**	**Main Additional Features**
DS1/RAX [157]	PS [110] EXEC Livingstone [227]	PAMGO	State Variables time-points Qualitative model Transition automata with probabilities and costs	Autonomous Spacecraft	Layered (RA)	Time management Resource management Diagnostic and recovery
IxTeT/PRS [108]	IxTeT [90] Exec [137] Reco [65] PRS [106] GenoM [76]	PAMO	State Variables, time-points Chronicles Procedures	Exploration & rescue robots	Layered (LAAS)	Time management Resource management Verification and validation (functional layer)
IDEA [155]	IDEA EUROPA [79]	PA	State Variables, intervals	Exploration robots	Agent-based (IDEA)	Time management Integrated knowledge representation Refinement with lookahead
T-REX	T-REX [147] EUROPA [79] APSI[81]	PA	State Variables, intervals	AUV Service and domestic robots	Teleo reactive	Time management Integrated knowledge representation Refinement with lookahead
RMPL[225]	Titan [226] Burton [228] Kirk [224] Drake [52] Pike [140]	PAM	Probabilistic Timed hierarchical automata Temporal Plan Networks Probabilistic TPNs, TPNs with Uncertainty	Exploration & rescue robots Autonomous Spacecraft	Specific	Time management Diagnostic and recovery Integrated knowledge representation Refinement with lookahead Handling of uncertainty
WITAS [63]	TALPlanner [133] DyKnow [101]	PAMO	Temporal Action Logic Stream, policy Chronicles	Autonomous aerial vehicles	Layered [62]	Time management Resource management Integrated knowledge representation Explicit handling of uncertainty Verification and validation
ASPEN/ CASPER	ASPEN [178] CASPER [43] TCA [200] TDL [201] Plexil [216]	PA	State Variables Intervals Programs	Exploration & rescue robots	Layered	Time management Resources management
XFRM	XFRM [16] RPL [15] SRC [14] SRP [17] CRAM [18]	PAL	RPL, CRAM Plan Language Prolog	Service and domestic robots	Open world (CRAM)	Integrated knowledge representation Refinement with lookahead Open models
Cypress [223]	Sipe [222] PRS [158] CPEF [159]	PAG	ACT Formalism	Military		Time management Resource management Integrated knowledge representation Refinement with lookahead

Table 2: Analyzed systems, clustered into approaches and associated components, with their main deliberation functions (P: planning, A: acting, M: monitoring, O: observing, G: goal reasoning, L: learning) and other features.

Numerous contributions fit into the scope of this survey. A uniform level of discussion of all of them would be too long and too shallow. To present a meaningful view of the entire field ("perceiving the forest from the trees") we organized the surveyed systems into the following three clusters:

	Deliberation Functions	Knowledge Representation	Robotics Domains	Additional Main Features
SSP [146]	PA	Factorized MDP		Handling of uncertainty
MDP [29]	PA	Factorized MDP		Handling of uncertainty
MAPL [31]	PA	State Variables Belief Variables Assertions	Service robots	Multi-robot integration
DL/HTN [96]	PA	DL and HTN	Service robots	
RACE [186]	P	HTN	Service and domestic robots	Open models
Petri-Net [221]	A	Petri Net		
SMACH [24]	A	Automata	Service robots	ROS
Readylog [71]	PA	Situation Calculus		Logical Framework
Pearl [171]	PA	MDP	Service robots	Handling of uncertainty
Ressac [207]	PA	MDP	UAV	Handling of uncertainty
Coradeschi [53]	PO	Perceptual anchoring	Service robots, UAV	Tracking and reacquiring anchors integrated to planning
PTL [115]	PO	Ambigous anchors with conditional planning in belief space	Service robots	Partial matching properties - features, definite and indefinite descriptions
HiPPo [203]	PO	Hierarchical POMDP	Object manipulation	Sensor and manipulation planning
Velez [214]	PO	Bayes network	Service robots	Observation planning and navigation
Elvira [84]	POL	Semantic Maps	Service robots	Hierarchical environment models
Astrid [83]	POL	Semantic Maps	Service robots	Integration to planning & acting

Table 3: Discussed systems.

- *Analyzed systems* (Table 2): these systems or groups of systems cover several deliberation functions; they often span over a long line of contributions, implementations and experiments that we group together; they are presented with some level of details often in several of the following sections.
- *Discussed systems* (Table 3): these systems are more focused on a single or a few contributions; they often have narrower integration scope and experimental validation; they will be discussed briefly in a single section.
- *Cited systems*: papers in this cluster bring interesting ideas or techniques, relevant to deliberation in robotics, but they have not been necessarily integrated or experimented with on a robot, or seem to be too close to other systems in the previous two categories to be referred to along with the discussion of other systems.

It is important to underline that no ranking is intended through this clustering, which is introduced solely for the commodity of the presentation, to help surveying a broad field into a readable picture. Tables 2 and 3 summarize the main characteristics of the analyzed and discussed systems: the deliberation functions integrated, their knowledge representations, their type of robotics applications, the architectures used and additional features when relevant.

In what follows, we'll review contributions that have an emphasis on, successively, planning, acting, monitoring, observing and modeling (Sec. 3 to 7). In each of these sections, an introduction gives an overview of the corresponding function within integrated deliberation systems; a summary outlines the main features of surveyed work. A synthesis (Sec. 8) reviews the analyzed deliberation systems with respect to their main properties and application requirements.

3. Planning

Planning is a core deliberation function for an autonomous robot. It can be viewed as the coupling of a *prediction problem* and a *search problem*. The former refers to the capability to predict the effects of an action in some context. The latter searches through all possibilities in order to choose feasible actions and to organize them into a plan, which is foreseen, at planning time, to be feasible and to achieve a desired goal or optimize a performance criterion.

Predictive models specific to a particular type of action can be quite precise. For example, physics models with good parameter identification and state observation systems can predict precisely the effects of an action. Conversely, abstract models of the form "precondition–effect" are general enough to represent a variety of tasks but quite shallow in their predictive capabilities. Usually, abstract models are easily specified and developed, while specific models demand significant engineering efforts. This is the issue of *domain-specific* versus *domain-independent* planning, for which there is no final tradeoff.[1] The realistic option is to integrate both: domain-specific planning for the essential actions of a robot, i.e., mobility and manipulation, and domain-independent task planning to handle adaptation to a variety of tasks at a more abstract level. A few approaches, surveyed next, have been proposed to make this integration. Some models acquisition approaches (Sec. 7) further extend the scope of domain-specific techniques with reduced engineering efforts. Two considerations are particularly relevant in robotics:

- *Plan revision, repair and reuse*: these planning modalities are often more important in practice than the synthesis of new plans from an empty agenda. Complexity considerations in classical task planning would favor replanning to repair and reuse [160]. But this is not supported by other considerations such as acting costs, ongoing commitments of a robot and plan stability issues [77, 210, 212].
- *Simulation and sampling*: "precondition–effect" representation of actions tends to be shallow. Simulation systems of robots and environments, when available, can be used to achieve reliable predictions, in particular with sampling techniques. This is illustrated in the integration of motion planning to task planning, e.g., [33, 4], in probabilistic planning techniques, and in reinforcement learning (Sec. 7.2).

Several other options arise for the design of a robot planner:

- *open-loop* versus *closed-loop* plans: the latter are conditional plans involving perception (i.e., do this, then, depending on the perceived state of the world, do that or this), while the former are unconditional plans. In the open-loop case, the designer is generally aware that the predictions of a deterministic model and the assumption of a static environment do not hold. But the designer trusts the monitoring and acting functions to deal with and correct small variations between predicted and observed states, and replanning to handle more significant ones.
- plan *horizon*: it can be an *indefinite* horizon of a one-shot plan that achieves the objectives, a *receding* horizon partial plan that will be further progressed while acting, or an *infinite* horizon policy that optimizes the robot behavior. In many cases, the application domain determines the choice. For example, an infinite horizon approach is adequate for maintaining a property in a dynamic environment, e.g., keeping a room clean; while a moving target mission is better addressed with a receding horizon method.
- *deterministic* versus *nondeterministic* models: the latter predict alternative effects of an action and, in the probabilistic case, take into account their likelihood. Nondeterministic planning generates policies, which are conditional closed-loop plans.

Over the past decades, domain-independent task planning has achieved tremendous progress, mostly on the search problem while keeping the prediction problem quite simple with the precondition–effect model. A speed up of a few orders of magnitude in the performance of classical planners has been achieved; numerous extensions to relational representations and improvements in algorithms for non-deterministic, probabilistic and other non-classical planning have been developed. A survey of task planning is not the purpose of this section (see [86, 91, 222, 234]). We will focus instead on problems and approaches for the integration of planning in a robot deliberation system. In addition to the integration of motion and manipulation planning to task planning, robotics stresses particular issues such as handling time and resources, uncertainty, partial knowledge and open domains. The rest of the section surveys a few approaches to these problems, summarized in Table 4.

3.1. Motion and Task Planning

Robots have to plan precise movements of their wheels, legs, arms, and joints. Motion planning and task planning are different problems that use distinct representations and search techniques. The latter has been mostly considered by the AI community, while the former has been studied by the robotics and computational geometry communities. In simple cases one can decouple the two problems: task planning produces a high level plan whose steps requiring motions can be handled by a specific motion planner. Most of the systems in Table 2 embed a motion planner used in

[1] In the terminology of task planning, a motion planner is domain-specific even if it is applicable to a wide diversity of robots and environments.

Approaches	Systems	Open or Closed Loop	Horizon	Nondeterminism	Concurrency	Solution Plan	Knowledge Representation	Search Algorithm	Integration with other deliberation functions
Temporal planning	IxTeT [89]	Open	Indefinite	Deterministic	yes	Flexible	State variables, time-points	Temporal in plan space	Acting with IxTeT-eXeC monitoring with IxTeT-Reco
	PS [110]	Open	Indefinite	Deterministic	yes	Flexible	State variables, intervals	Temporal in plan space	Acting, monitoring and goal reasoning
	EUROPA [79]	Open	Indefinite	Deterministic	yes	Flexible	State variables, intervals	Temporal in plan space	Acting (see IDEA & T-REX)
	APSI [81]	Open	Indefinite	Deterministic	yes	Flexible	State variables, intervals	Temporal in plan space	Acting with T-REX
	ASPEN [178]	Open	Indefinite	Deterministic	yes	Completely instantiated	State variables, intervals	Local search	Acting (see CASPER)
Probabilistic Planning	SSP [146]	Closed	Indefinite	Nondeterministic	no	Partial policy	MDP	Heuristic and sampling	Acting
	RFF [206]	Closed	Receding	Nondeterministic	no	Partial policy	MDP	Heuristic	Acting
	MCP [141]	Closed		Mixed	no	Partial policy	Factorized MDP	Compressed deterministic part	Acting
Open World Planning	MAPL [31]	Closed	Indefinite	Delayed	no	Assertions to be refined	State variables, belief variables, assertions	Classical	Mixed planning/acting
	Hartanto [96]	Open	Indefinite	Deterministic	no	Completely instantiated	HTN, DL	DL reasoning, HTN	
	Race [186]	Open	Indefinite	Deterministic	no	Completely instanciated	HTN	HTN	Communication using BlackBoard

Table 4: Surveyed planning systems: main features, knowledge representations, algorithms and links with other deliberation functions.

this decoupled approach. When the mission is more complex or the environment more constrained, this decomposition is no longer possible: the motion constraints must be taken into account early in the task planning phase. Let us briefly present the state of the art in motion and manipulation planning and review a few approaches (listed in Table 5) integrating it with task planning.

	Feasible motions		
Search	Planned	Estimated	Both
State space	Asymov [36] CPMP [44] I-TMP* [97] CTAMP [22]	SamplSGD [238]	
Hierarchical		AHP [231]	Kaelbling [113]

Table 5: Integrated motion and task planning, (*) with online planning and acting

Motion and manipulation planning are key capabilities for a robot, requiring specific representations for geometry, kinematics and dynamics. *Probabilistic Roadmaps* (PRM) and *Rapidly-Exploring Random Trees* (RRT) are well developed and mature techniques for motion planners that scale up efficiently and allow for numerous extensions [136]. The basic idea is to randomly sample the configuration space (*C*-space) of the robot (i.e., the vector space of the robot's kinematics parameters) into a graph where each vertex is a point in the space denoted C_f of free configurations (away from obstacles), and each edge a direct link in the free space between two free configurations. Initial and goal configurations are added to this graph, between which a path is computed. This path is then transformed into a smooth trajectory. Manipulation planning requires finding feasible sequences of grasping positions, each of which is a partial constraint on the robot configuration that changes its kinematics [198]. Many other open problems remain in motion and manipulation planning, such as dynamics and stability constraints, e.g., for a humanoid robot [114], or visibility constraints, e.g., to allow for visual servoing [41].

Task planning and motion/manipulation planning have been brought together in a few approaches. The CPMP planner [44] builds a task planner domain from a tree constructed by a motion planner in the robot *C*-space. It then plans symbolically with a state space planner and converts back the found plans into the *C*-space.

The Asymov planner [36] combines a state-space task planner with a search in the motion planner configuration space. It defines *places* which are both a state in the task planner space, and cover a range of free configurations in C_f. These places define bridges between the two search spaces. These two spaces are not explicitly constructed, but for each generated state of the task plan, the system checks that at least one configuration C_f is reachable. The state-space search prunes a state whose corresponding set of free configurations does not meet current reachability conditions. This approach has also been extended to manipulation planning and to multi-robot planning for joint activities such as two robots assembling a table in a cluttered environment.

The integration of motion to task planning is also explored in [231] with Angelic Hierarchical Planning (AHP). AHP plans over sets of states with the notion of *reachable sets of states*. These sets are not computed exactly, but bounded, e.g., by a subset and a superset, or by an upper and a lower bound cost function. These costs are obtained by simulations of the basic activities, including motion activities, for random values of the state variables. A high-level task has several possible decompositions into primitives. The set of possible decompositions of a plan containing several high-level tasks is the Cartesien product of all possible decompositions of its actions. It is not necessary to ensure that all decompositions are feasible. A plan is acceptable if it has at least one feasible decomposition. Decomposition of a task can be postponed, e.g., until the acting time for the achievement of that task. At that point, the robot chooses a feasible decomposition (hence the reference to an angelic semantics of nondeterminism).

A different coupling of a hierarchical task planner to *fast geometric suggesters* is developed in [113]. These suggesters are triggered when the search in the decomposition tree requires geometric information. They do not completely solve the geometric problem, but they provide information that allows the symbolic search to continue up to a leaf of the tree. The system alternates between symbolic planning phases and the real execution of geometric actions in the environment. Planning proceeds while acting. This allows the task planner to call the geometric planner (not the suggesters) in a fully known state. Thus, this approach relies on two strong assumptions: the geometric preconditions of the tasks can be calculated quickly and efficiently (by the suggesters), the subgoals resulting from a task decomposition must be executed in sequence (no parallelism). The authors acknowledge that their system is not complete, and that the subgoals sequencing assumption requires actions to be reversible at a reasonable cost. Yet, for problems where these assumptions are satisfied, the system is able to quickly produce correct plans.

The geometric backtracking approach of [22] and subsequent work on a system called CTAMP relies on the automated synthesis of geometric constraints from the task plan and their propagation, as linear programs, to filter out samples in the configuration space. Remaining configurations, if any, are further searched with respect to sufficient conditions of inverse kinematics, collision detection and motion planning to make sure that each step specifies collision free and kinematically achievable poses with respect to the previous ones. The approach has been developed for manipulation problems and experimented with the dual arm Justin robot.

PRM and RRT motion planners used in cited approaches solve feasible queries very efficiently, but they handle infeasible queries with a cutoff time that can be a serious drawback in constrained environments. The incremental algorithm I-TMP [97] integrates closely task and motion planning with adaptive allocation of computational time where it can be most useful. It has been tested on multi leg robots navigating on rough terrain.

Let us also mention the physics-based technique of [238] which couples a physics engine simulator with RRT-like search methods for planning the interactions of a robot with non actuated bodies with varying degrees of controllability

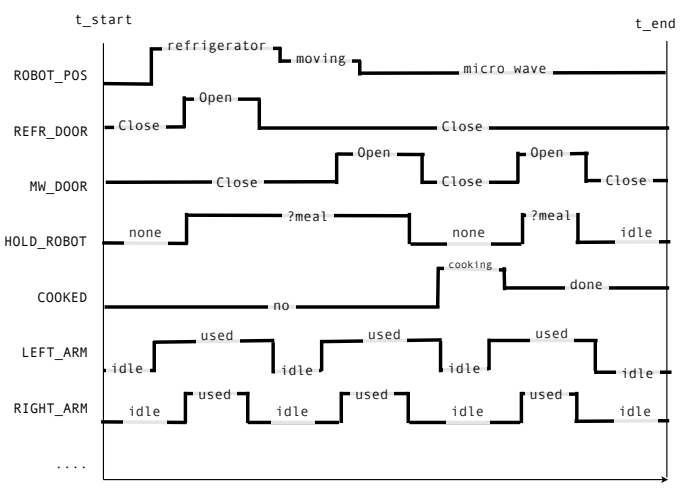
in order to reach some goals. The approach was tested in minigolf and robot soccer domains. Similarly, the SamplSGD system [172] adds differential constraints to the symbolic and geometric specifications of a planning domain. It relies on sampling techniques to selectively explore the continuous configuration and dynamics space. Task planning provides discrete actions and regions of the continuous space that sampling based motion planning can explore. The two planners interact through estimates on the utility of each action, which are computed by reinforcement learning techniques. SamplSGD was tested on simulated manipulation tasks.

3.2. Temporal Planning

Most robotics applications require explicit time models to handle durative actions, concurrency, synchronization with exogenous events and other robots, and temporal constraints, possibly with flexible lower and upper bounds.

Various approaches manage explicit time representations for planning. Several planners extend state-based representations with durative actions (e.g., TGP [202], VHPOP [237] or Crickey [48]). A few of them can manage concurrency, or even linear continuous change (COLIN [50]), or can plan and repair in real-time [188] . However, temporal planners that rely on *timelines*, i.e., partially specified evolution of state variables over time, are more flexible in the integration of planning and acting and, in most cases, more expressive, e.g., allowing the reference to instants beyond starting and ending time of actions. These planners implement plan-space search algorithms more often then state-space techniques. Their usual representation ingredients are:

- temporal primitives: time-points or intervals (tokens),
- state variables, possibly parametrized, e.g., position(object32), and rigid relations, e.g., connected(loc3, room5), invariant over time,
- assertions on the persistence of the value of a state variable over time, or the discrete or continuous change of values,
- temporal constraints: Allen interval relations [5] or Simple Temporal Networks [58] over time-points,
- atemporal constraints on the values and parameters of state-variables.




```
task FIX_DINER(?meal)(t_start,t_end){
  ?meal in MEALS;

    hold(ROBOT_POS():refrigerator,(t_start,t_re));
    event(REFR_DOOR():(close,open),t_ros);
    hold(REFR_DOOR():open, (t_ros,t_roe));
    event(REFR_DOOR():(open,close),t_roe);

    hold(HOLD_ROBOT():?meal, (t_shm,t_ehm));

    hold(ROBOT_POS():micro_wave, (t_mws,t_mwe));
    event(MW_DOOR():(close,open),t_mwo);
    hold(MW_DOOR():open, (t_mwos,t_mwoe));
    event(MW_DOOR():(open,close),t_mwoe);

    hold(MEAL_IN_MW():?meal, (t_smimw,t_emimw));

    event(COOKED(?meal):(no,cooking),t_cs);
    hold(COOKED(?meal):cooking,(t_cs,t_ce));
    event(COOKED(meal):(cooking,done),t_cs);

    ....

    use(LEFT_ARM():1,(t_start,t_end));
    use(RIGHT_ARM():1,(t_start,t_end));

    (t_end - t_start) in ]0,600];
}nonPreemptive
```

Figure 4: Example of an IxTeT chronicle modeling an action for a service robot.

The chronicles of the IxTeT planner [89] illustrate such a representation. A chronicle defines time-points, temporal constraints between its time-points, changes in the values of state variables (denoted *event*), persistence of these values over time (denoted *hold*), and atemporal constraints over state variables parameters and values (see Fig. 4).

Other planners such as PS, the DS1/RAX planner [110], EUROPA [79] and APSI [81], rely on a similar representation with timelines and tokens representing changes and persistences of state variables values over time intervals. Some of these timelines are directly connected to actions and percepts (to integrate perception). Temporal constraints in planning operators use the symbolic interval algebra [5] (see Fig. 5). Simple atemporal constraints over state variables are also allowed. Goals are expressed as an *unexplained* assertions, i.e., required state-variable values to be

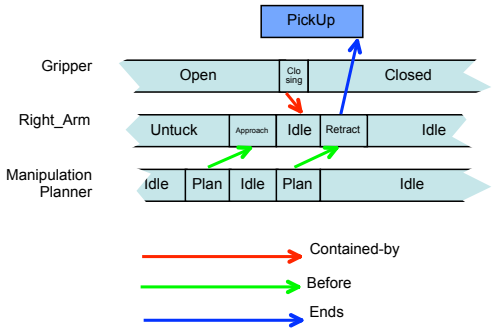


Figure 5: Model of a PickUp action with three state variables: Gripper, Right_Arm, Manipulation Planner.

accounted for by the planner through actions. Preconditions and predicted effects are expressed as assertions constrained with respect to the duration of actions.

Planning proceeds in the plan-space by detecting flaws, i.e., unexplained changes and possible inconsistencies, and repairing them through additional actions and constraints. It makes use of various heuristics, constraint propagation and backtrack mechanisms. It produces partially specified plans, that have no more flaws but still contain constrained but non-instantiated temporal and atemporal variables. This *least commitment* approach allows for some flexibility to adapt to the execution context. Some planners can handle bounded planning time and take it into account together with the execution horizon during the planning process.

3.3. Probabilistic Planning

Markov Decision Processes (MDPs) offer a classical framework for planning with uncertainties and probabilistic effects. A planner computes an optimal policy, i.e., a universal plan mapping states into applicable actions. The classical framework considers infinite horizon problems, e.g., to maintain a property in dynamic environment. It can be extended to partially observable systems (POMDPs). It is effective as long as the state space and the cost and probability parameters can be entirely acquired and made explicite, and, for POMDPs, remains of small size.[2] However, most deliberation problems in robotics do not allow for an explicit enumeration of their state space, and hence cannot afford a universal plan. Fortunately, many problems are focused on reaching a goal from some initial state. Heuristic search algorithms for Stochastic Shortest Path (SSP) problems [146] with factorized and hierarchical representations [29] address this class of problems. They offer probabilistic approaches to robotics deliberation that still need to be developed and deployed on robots, beyond classical test problems.

An SSP problem focuses on a *partial policy*, *closed* for the initial state s_0 (i.e., a policy π defined on all states reachable from s_0 by π), terminating at goal states. An SSP is the equivalent for And/Or graphs to path search problems in deterministic graphs. For very large implicit search spaces with sparse models (few applicable actions per state, few nondeterministic effects per action, including deterministic actions), a significant scaling up with respect to classical dynamic programming methods can be achieved with heuristics and sampling techniques [146].

Most heuristic search algorithms for SSPs are based on a two steps schema, called Find&Revise: *(i)* Find a pending state s among the descendant of s_0 reachable with current policy, and *(ii)* Revise the estimated value and policy of s with the so-called Bellman update. A state s is pending when it needs to be expanded or when its value and policy need to be updated towards the best (or good) policy from s. That framework can be implemented in different ways, e.g., with a best-first search, as in AO*, LAO* [95] and their extensions, with a depth-first iterative deepening search, as in LDFS [27], or with a random walk search, as in RTDP [13], LRTDP [26] and their extensions.

These algorithms assume SSP problems with a *proper policy* closed for s_0 (i.e., one that reaches a goal with probability 1), and any improper policy has infinite cost. A generalization of SSPs to relax this last assumption allows to seek a policy that maximizes the probability of reaching a goal state, a very useful and desirable criteria [124]. Other issues, such as dead-ends (states from which it is not possible to reach a goal) have to be taken care of, in particular in

[2] An MDP with n k-ary state variables has a state space in k^n; a POMDP is an MDP on the belief space whose discritized size is in 2^{k^n}.

critical domains [122]. Heuristic search algorithms in SSPs are more scalable than dynamic programming, but they still cannot address large domains, unless these domains are carefully engineered and decomposed.

Various sampling and approximation techniques offer alternative approaches to further scale up probabilistic planning. Among these approaches, determinization techniques transform each non-deterministic action into several deterministic ones (the most likely or all possible ones), then it plans deterministically with these actions, online and/or offline. For example, SSiP [211] combines replanning when needed with planning over a parametrized horizon of at least k steps to be guaranteed without replanning. RFF [206] generates an initial deterministic plan, then it considers a fringe state along a non-deterministic branch of that plan and, as long as the probability to reach that state is above some threshold, it extends it with another deterministic plan toward a goal or an already solved state. Similar ideas are developed in sampling approaches. One of their advantages is the capability to work without a priori estimates of the probability distributions, as long as the sampling is drawn from these same distributions. Several strategies have been designed for the control of the search, e.g., sample more sparsely at the deeper level of the look-ahead tree. Bounds on the approximation quality and the complexity of the search have been obtained, with good results on various adaptations and extensions of the LRTDP and UCT algorithms, e.g., [117, 35, 123].

Although MDPs are often used in robotics at the sensory-motor level, in particular within reinforcement learning approaches (Sec. 7.2), SSP techniques are not as widely disseminated at the deliberative planning and acting level. An interesting example is the sparse MDP planner of [141] that couples SSP with classical planning: when most actions are deterministic but of a few that are probabilistic, it searches through and reduces the deterministic parts of the planning space into a compressed MDP. The approach has been tested in a robotics exploration problem.

3.4. Planning in Open Domains

Most classical planners rely on the *closed-world assumption*: a fact not explicitly stated is false.[3] This assumption allows for a concise description of states and efficient computation of state-transitions. But it is too restrictive and seldom acceptable in robotics. A few techniques relax this assumption, e.g., planning with state-variables and timelines allows for partially specified instances of a domain. But other issues remain open, among them the following:

- Planning with respect to domain objects and properties that are unknown when planning starts but that can be discovered at acting time, in particular through planned information gathering actions. These new facts will be used to further refine the remainder of the plan.
- Allow the planner to query databases for facts it needs specifically to address a given planning problem, or to query knowledge bases for additional models of its environment that are relevant to the task at hand.

The former issue of planning with information gathering is studied by several authors with conditional plan approaches, as in the PKS [168] system. The continual planning approach of MAPL [31] postpones part of the planning process. In its place, it introduces information gathering actions to later develop the missing parts of the plan. To this end, the planner uses assertions that abstract actions to be refined later, after information gathering. This mixed planning/acting approach is well adapted to dynamic environments where planning for subgoals that depend of yet unknown states can be delayed until the required information is available (through properly planned information gathering actions). The approach has been extended to multi-robot distributed planning/acting. It has been tested in simulated navigation problems.

The latter issue of acquiring additional models and data at planning time is being addressed by a growing number of projects. The driving idea is to enrich a robot deliberation system with functionalities akin to *semantic web*, using a *domain ontology* associated to some reasoning capabilities. For example, the ObjectEval system of [190] acquires from the web statistics about possible locations of objects of different classes. It uses them in a utility function for finding and delivering objects in an office environment. Other approaches use Description Logic (DL), a fragment of first-order logic, to handle statements about objects, properties, relations and their instances with inference algorithms for querying large stores of data and models over the web. Most implementations rely on OWL, the standard Web Ontology Language. OWL handles an open-world representation where facts can be *true, false* or *unknown*.

The approach of [96] couples DL with an HTN planner. A robot requires a large store of knowledge about its environment, only a small part of which is relevant to a given planning problem. DL allows the planner to synthesize a compact equivalent HTN problem focused on this relevant part. The corresponding ontology includes HTN planning

[3] Some take the less restrictive view of "negation by failure": a fact not entailed from explicit statements is false.

concepts (i.e., tasks, methods, states, etc.) as well as domain concepts. The approach was tested on navigation problems. The RACE project [186] designs a service robot with learning capabilities whose entire architecture is built around an open DL database, akin to a *"Blackboard"* architecture [100]. This feature is used to combine data about low-level robot activities, needed in particular for learning, with higher-level semantic descriptions provided by the ontology. The system supports a PR2 platform performing the tasks of a waiter.

The Open Robot Ontology (ORO) system of [138] was motivated more by extending the robot's knowledge base then by improving the performance of its planners. ORO is built with the same OWL representation and reasoner as the previous systems. It offers queries and updates of a knowledge base about the environment. ORO services are used by various components of the robot, in particular by a situated dialogue management system and a planner. ORO has been tested with a few platforms for service robotics tasks (illustrated in Fig. 2(c)).

The RoboEarth and KnowRob projects [218, 208] aim at allowing robots having different platforms to share and reuse knowledge over the network for the purpose of performing new tasks and speeding up learning and adaptation. An OWL open source library stores shared models of objects (e.g., images, CAD models), environments (e.g., maps and object locations), and actions (e.g., action recipes and skills) together with their relations and properties in a general ontology. Each robot is endowed with means for querying and updating this database, as well as adapting its models. The concept was tested on a few demonstrators for exploring a maze using different platforms sharing maps and path information, for serving a drink or opening drawers without a priori knowledge of the environment setting.

3.5. Summary

Motion planners are essential in robotics; they are available in most of the surveyed systems. Task planning is also needed for deliberation. The assumption of independence between task and motion planning does not hold in tightly coupled problems. The state of the art for integrating task planning to motion planning, and more generally to other domain-specific planners, is progressing significantly.

In addition to space, time plays an important role in planning; it allows for concurrent activities and synchronization. Most of the task planners in our analyzed systems take into account time, often with an explicit representation with state variables over timelines. Constraint programming and plan space search are also quite popular in reviewed systems. As we will see next, they offer some advantages in the integration of planning and acting and in plan repair.

The state of the art in nondeterministic and probabilistic planning has progressed. These approaches are appealing for addressing the uncertainty in knowledge and world variability. For the moment, they are mostly developed at the sensory-motor level and, to some extent, at the acting level (see Sec. 4.6). The bottleneck for their deployment in deliberation planning mainly is the modeling effort and the complication of integrating space and time.

We described planning as the coupling of a prediction problem and a search problem. Significant research efforts are devoted to the latter. More investigation in the former is needed for improving the predictive capabilities of models used by planners. Simulation-based planning appears to be a promising option. Open domain planning also has a large potential and should be more widely deployed in robotics deliberation systems.

4. Acting

Planning is easily specified as an offline predictive function, decoupled from the intricacies of the executing platform. In contrast, acting is more difficult to define as a deliberation function. The frequent reference to *execution control* is often too reductive. There is much more to it than triggering abstract actions prescribed by a plan. Acting has to handle noisy sensors, unreliable actuators and imperfect models. It requires *nondeterministic*, *partially observable* and *dynamic* environment models, dealt with through *closed-loop commands*. Acting has often a central role in deliberation. It has to carry out in particular the following functions:

- *Refinement*: plan steps (i.e., actions) given by a task planner are seldom directly executable as robot commands. An action like `open(door1)` is a primitive step for a planner but requires significant deliberation for its achievement. The refinement process is context dependent. It relies on lower-level models, that we called *skills*, appropriate for the context. Skills can be chosen and adapted from a library or synthesized online. The skill into which an action is refined may change during execution. For example several navigation skills may offer different localization or motion control capabilities adapted to different context; hence the `goto(room1)` action can be refined into a sequence of several navigation skills. Acting also has to consider alternative refinements in case of failure.

- *Reaction* to events: there are expected events, for plan synchronization purposes, or unexpected events. Events can be dealt with locally with appropriate skills. They may have negative or positive consequences for the current plan or even for the current goal. They may require plan repair in addition to the local reaction.
- *Instantiation*: unless synthesized online, acting skills are applicable to a range of situations. Their models can use non instantiated variables whose value can be chosen at execution time or observed in the environment. These values get propagated down to commands with respect to the perceived context.
- *Time management*: acting is performed online. The dynamics of the environment has to be dealt with. Adequate responses must be given in a timely manner. There are issues of time as a resource with deadlines, durations and deliberation time. There are also issues of rendezvous, synchronization with events, and time constraints.
- *Nondeterminism*: Applying an action may result in several possible states. The occurrence of exogenous events in a dynamic environment is seldom deterministically predictable. Further, not all relevant state variables are observable with certainty. We will see that some of the presented techniques can handle sources of nondeterminism.
- *Plan repair*: As introduced in Section 3, plan repair is clearly a planning activity. Yet, it is strongly linked to acting which triggers repair or replanning in the appropriate conditions with respect to committed execution. Acting has to support this feature.

The overlap between planning and acting leaves numerous design options open. A few authors develop a framework where both deliberation functions are handled in the same search space, with the same techniques. However, most approaches address the distinct requirements of planning and acting with decomposition scheme (Fig. 3) where planning reasons with abstract *descriptive models* of actions while acting uses *operational models* to refine actions into procedures, called *skills*, that are further refined into *commands*. This decomposition scheme may rely on distinct knowledge representations, e.g., STRIPS operators for the former combined with PRS [106] or RAP [75] procedures for the latter. In some cases a single representation is used for both planning and acting, e.g., in Golog [139], IDEA [155] and T-REX [147]. Other approaches use a single representation but at different levels of abstractions and refined appropriately, as in Hierarchical MDPs [98] for example.

Other important issues are how the acting skills are acquired, either synthesized automatically, hand written, or obtained through a learning mechanisms, and how they are used, as directly executable programs or as specification models from which further deliberation is performed. Acting knowledge has also to be checked for consistency with respect to planning models. Some skill formalisms are more amenable to validation and verification than others, hence more adapted to applications where criticality is high (see Table 1.)

Various computational techniques can be used to design a deliberate acting system. We propose to organize these approaches into five categories. Their main features are summarized in Table 6.

4.1. Direct mapping of plan steps into robot commands

This class of approaches assumes that planned steps are directly executable as commands by the platform. Often, there is an elaborate processing at the platform level, hard coded in the functional layer of the robot, but there is no explicit deliberation for action refinement. Acting is focused mostly on reacting and, in some cases monitoring.

Planex [73] is one of the earliest acting system, deployed on the Shakey robot. Plans produced by STRIPS are executed without any action refinement: Planex directly maps plan steps into commands. It assumes a correct and complete state update after each action is executed, from which it detects failures but also opportunities for pursuing the plan. It relies on *triangle tables* which have been advocated for learning and generalizing a collection of plans, and as well as for programming with a hierarchical collection of actions. [194] extended the idea with *universal plans* that define a class of plans applicable in a set of states leading to a particular goal. This class of approaches handles in a restricted way nondeterminism and repair, as it may adapt the plan when the observed state is not the nominal one.

One of the problems with these and similar approaches is their lack of robustness in the interaction with the physical world. They remain at a high level of abstraction, assuming full observability, no uncertainty on observations, yet they manage a limited form of nondeterminism due to exogenous events.

4.2. Acting with procedure-based approaches

In procedure-based approaches, action refinement is done with handwritten skills similar to imperative programs. RAP (Reactive Action Package) [75] is one of the earliest work on acting and reactive planning (i.e., without search). Each skill is in charge of satisfying a particular goal, corresponding to a planned action. Deliberation chooses the

		Functions					*Knowledge Representation*		
Approaches	**Systems**	**Refine-ment**	**Instan-tiation**	**Time han-dling**	**Nondet.**	**Repair**	**Specification**	**V&V**	**Shared P/A KR**
Direct	Planex [73]						Model		X
Procedure	RAP [75]	X	X				Program		
	PRS [106]	X	X				Program		
	Cypress/CPEF [223]	X	X				Program		ACT
	TCA/TDL [201]	X	X	X			Program		
	XFRM/RPL/SRP [16]	X	X			X	Program		X
Petri-Net	IMRS [221]	X		synchro.			Model	X	
	Procosa [12]	X		synchro.			Model	X	
	Petri-Net Plans [241]	X		synchro.			Model	X	
Automata	FSA [40]	X	X	X			Model		X
	PLEXIL [216]	X	X	X			Model		
	SMACH [24]	X	X	X			Model		
Logic	Golex [94]	X	X				Model	X	X
	ReadyLog [71]	X	X				Model	X	X
CSP	IxTeT-exec [137]	X	X	X		X	Model		X
	RMPL [105]	X	X	X	X	X	Model		X
	IDEA [74]	X	X	X		X	Model		X
	T-REX [180]	X	X	X		X	Model		X
	Casper [69]	X	X	X		X	Model		X
MDP	PEARL [171]				X		Model		
	K9 MDP [239]				X		Model		
	Robel [152]				X		Model		
	Ressac [207]				X		Model		

Table 6: Surveyed acting systems: main functions (*reaction* to events omitted since present in all systems) and knowledge representation.

appropriate skill according to the current context. The system commits to the goals it has to achieve, trying a different skill when one fails. RAP was interfaced with the AP [25] and PRODIGY [215] planners.

PRS (Procedural Reasoning System) [106] is a procedure-based action refinement and monitoring system. As in RAP, one writes skills (Operational Procedures) to achieve goals or react to particular events and observations. The system commits to goals and tries alternative skills when needed. Procedures share a database of facts and goals. PRS allows for concurrent procedure execution and multi-threading. Some planning capabilities were added to PRS [59] to anticipate execution paths leading to failure, by simulating the execution of procedures and exploring the different branches. A more advanced integration of a planner to PRS was later developed with IxTeT [137] (Sec. 4.5). PRS is used in numerous robotics platforms. PRS skills refine actions into lower level commands (e.g., ROS "actions" for a PR2). Procedure steps are subgoals, possibly executed concurrently, within conditional and control structures, monitoring and recovery instructions. When integrated with GenoM (a generator of sensory-motor modules [76]), PRS procedures handling the robot commands are generated automatically [108].

TCA [199, 200] was initially developed to handle concurrent planning and acting. It provides a hierarchical task decomposition framework, with explicit temporal constraints to allow tasks synchronization. Task decomposition integrates specific geometrical and motion planning (e.g., gait and footfall planning for the Ambler robot). The Task Definition Language (TDL) [201] extends TCA with a wide range of synchronization constructs between tasks. It focuses on task execution and relies on systems like Casper/Aspen for planning.

XFRM [16] illustrates another approach which uses transformation rules to modify hand written plans expressed in the Reactive Plan Language (RPL). Unlike the above systems, it searches in plan space to improve its skills, using simulation and probabilities of possible outcomes. It replaces the currently executed plan on the fly if it finds another one more adapted to the current situation. This approach evolved toward Structured Reactive Controllers (SRC) and Structure Reactive Plan (SRP) [14], but still retains the XFRM technique to perform planning using transformation rules on SRP. It has been deployed on several prototype service robots.

Other similar procedure based approaches have been successfully deployed, such as IPEM [7], EXEC on DS1/RAX [157], PRS-Lite [158], which offers a richer goal semantics than the regular PRS, Cypress [223] and

CPEF (Continuous Planning and Execution Framework) [159], which combines PRS and Sipe [222].

Most procedure-based approaches focus on the refinement and instantiation functions. They fill the gap between plan steps and robot commands. Skills allow for control structures (conditional, loop, recursion) and are hand written; they are easy to use and deploy, but difficult to formally validate or verify. But of TCA/TDL, which offers synchronization mechanisms, these approaches do not handle temporal reasoning but can offer real-time capabilities.

4.3. Acting with automata and Petri net approaches

Robots integrate several concurrent sensory-motor processes that require strong and well defined synchronization and sequencing. Quite naturally, Petri nets have been used for that in a few experiments. In [221] Petri nets model the various functional components, the proper order of execution among them as well as the required coordination. The model can be used in simulation mode for verification and performance testing. Similarly, [12] proposes to use Petri nets to model hierarchically the supervision procedures of a UAV. It provides a verification tool to check for deadlocks and unwanted loops in the model. In [241], the authors propose Petri net plans for high level programming of multi-robot systems. They validate some properties of the system with reachability analysis and deadlock search.

It seems quite natural to refine an abstract action into a graph of elementary states and label the edges with sensory-motor signals and commands. Finite State Automata (FSA) have been studied as acting models. For example, FSA have been used jointly with IxTeT in [40]. Each IxTeT planned action has a corresponding FSA written by the programmer which specify the different states the action can be in. Acting controls the transitions between the states of the FSA according to external or internal events, and checks that they are consistent with the plan being executed. These transitions trigger lower level actions which are refined using scripts and executed within an activity tree.

PLEXIL, a language for the execution of plans, illustrates a representation where the user specifies nodes as computational abstractions [216]. A node can monitor events, execute commands or assign values to variables; it may refer hierarchically to a list of lower level nodes. Execution can be controlled by a number of constraints (start, end), guards (invariant) and conditions. PLEXIL focuses on acting. But of the input plan, it and does not share data with the planner. It has been developed for space applications and used with planners such as CASPER.

SMACH, the ROS execution system, also implements an automata-based approach [24]. The user writes a set of hierarchical state machines. Each state corresponds to the execution of a particular component of the robot. ROS actions,[4] services and topics are associated to the state; execution proceeds to the appropriate state according to returned values. The global state of the robot corresponds to the Cartesian product of the states of each components. Acting knows exactly in which state the system is, which eases deployment and monitoring. The interface with ROS actions, services and topics is very natural, but the semantics of constructs available in SMACH is limited for reasoning on goals and states.

All Petri net and Automata approaches address the functions of refinement, instantiation and time management (invariant constructs in the terminology of automata). Cited systems do not handle explicit time, nondeterminism, or plan repair (apart from local failure management). All rely on hand specified skills. Unlike the Procedure-based approaches, Automata and Petri net approaches allow for formal analysis (e.g., reachability and dead locks), which can be critical for the specification and the verification of acting models.

4.4. Logic-based approaches

A few systems try to overcome the tedious engineering bottleneck of detailed hand specification of skills by relying on logic inference mechanisms for extending high-level specifications. Typical examples are the Temporal Action Logic (TAL) approach (to which we will come back in Sec. 5) and the situation calculus approach. The latter is exemplified in GOLEX [94, 34], an execution system for the GOLOG planner. GOLEX was deployed on museum personal tour guide robot, and a robot serving coffee in an office environment.

In GOLOG and GOLEX the user specifies respectively planning and acting knowledge in the situation calculus representation. GOLEX provides Prolog hand programmed "`exec`" clauses which explicitly define the sequence of commands a robot has to execute. It also provides monitoring primitives to check the effects of executed actions. GOLEX executes the plan produced by GOLOG but even if the two systems rely on the same logic programming representation, they remain completely separated, limiting the interleaving of planning and acting.

[4] ROS "actions" on a PR2 correspond to platform commands in our terminology.

The PLATAS system [46] relies on GOLOG with a mapping between the PDDL langage and the Situation Calculus. It either uses GOLOG's internal iterative deepening planner or an external PDDL planner to produce a plan which is then executed by GOLEX. The READYLOG language (Real-time and dynamic GOLOG) [71], a derivative of GOLOG, combines planning with programming. Instead of planning with GOLOG, it provides a decision-theoretic planner which is used by the acting component when a problem needs to be solved for a given horizon (a short horizon in the Robotcup, given environment dynamics). The acting component monitors and perceives the environment through passive sensing, and acts or plans accordingly. ReadyLog was deployed on a robot-cup soccer robot and a service robot.

Overall these approaches perform refinement through the recursive application and instantiation of Prolog clauses. GOLEX offers no particular construct to handle time, nondeterminism, or plan repair. READYLOG goes further in handling nondeterminism and time (with its limited horizon search). From the knowledge representation point of view, GOLOG and GOLEX rely on the same language, but they remain clearly separated, GOLOG producing a sequence of "do" which is mapped to a sequence of "exec" in GOLEX. READYLOG is programmed in GOLOG. To our knowledge, none of these systems exploits the logic representation for performing consistency verification of planning and acting models; their benefit in terms of easier engineering specifications has not been documented.

4.5. CSP-based approaches

As seen in Section 3.2 temporal planning often produces partially instantiated plans with a *least commitment* approach allowing acting to adapt to the contingencies of the execution context. This has led to several developments of these approaches for acting. All time-based acting systems are de facto CSP-based.

Acting proceeds by progressively instantiating atemporal and temporal variables in the plan, through observation and context dependent choices. These values are propagated over the remaining constraints with consistency checking. Failures, that include observable but non-controllable variables (e.g., ending times of commands) outside of their assumed bounds, trigger plan repair.

IxTeT [89] was extended with execution capabilities following this approach [137]. Planning and acting share the partial plan structure that is dynamically produced during the planning phase and executed and repaired during the acting phase. An execution failure is seen as a new flaw in the partial plan space. The repair process tries to stay as close as possible to the rest of the plan. Repair requires invalidating part of the current plan and relaxing some constraints. This is done by memorizing for each causal link the reason why that link was inserted. Causal links associated with tasks are not removed from the flawed plan. If the repair does not succeed within some allocated time, the current plan is discarded and planning is restarted from the current situation. To bridge the gap with robot commands and perceptions, PRS is used jointly with IxTeT for refinement and skill execution. This system was deployed on the Dala robot for planetary exploration missions.

DS1/RAX implements a procedure-based acting approach; its successful deployment on the DS1 mission inspired the development of two planning & acting systems: IDEA and RMPL. IDEA [155] relies on a distributed approach where planning and acting use the same representation and differ only in their prediction horizon and allocated computing time to find a solution. The system is distributed over a hierarchy of components (agents), each being in charge of a particular function: e.g., mission planning, robot navigation, robot manipulation, payload management, etc; each has its own set of timelines, planning horizon, and computing time quantum. Components use the EUROPA planner to perform the constraint-based temporal planning. Components may share some timelines, accessed and modified by both, possibly with priorities. The timeline sharing mechanism allows the propagation of the planning results down to commands and, similarly, the integration from percepts to observations. In principle the hierarchy of components is able to express a continuum from planning operators down to commands. However, the scaling up of the technique proved to be difficult; the shortest computing time quantum that could be achieved was on the order of a second, not fast enough for the command level. Hence, that system had also to resort to hand programmed skills. Furthermore, the specification and debugging of action models distributed over several components proved to be complex and tedious.

IDEA has been experimented with on several platforms such as the K9 and Gromit rovers [74]. It led to the development of the T-REX system, deployed at MBARI for controlling AUVs [176, 180, 181] over long missions. T-REX is distributed, like IDEA, but differently organized into "reactors". Timeline sharing is used in both systems, but T-REX components are organized in such a way that constraint propagation is guaranteed to terminate (which is not the case in IDEA). T-REX is planner independent. MBARI implementation uses EUROPA with learned policies [142]. T-REX was also used with APSI [38], but not at the level of integration reached with EUROPA.

Casper [178, 119, 69] is a temporal constraint-based executor for the ASPEN planner that continuously produces fully instantiated plans. The resulting plan can be brittle and execution may often fail. However, the system relies on fast local search mechanisms (iterative repair) to reestablish plan consistency and resume execution. The system has been used on test-beds for exploration rovers and onboard real orbiting satellites.

Controllability is an issue for CSP-based approaches. It is due to contingent constraints and time points. The latter are for example the occurrence of an exogenous event or the end time of an action; they are not under the actor's control. A contingent temporal constraint expresses bounds on the uncertainty of a random variable duration; it cannot be reduced through constraint propagation, as usual in CSP. Simple Temporal Networks with Uncertainty (STNU) formalize these notions with a partition of their time points and constraints into controllable and contingent ones [217]. An STNU is *dynamically controllable* if there is a procedure for assigning dynamically values to controllable variables such that all the controllable constraints are met, given that at each step of this procedure, the values of contingent variables preceding that step are known and fit the bounds in the contingent constraints. Dynamic controllability can be checked in polynomial time [154, 153], including incrementally [165].

This property is maintained by a few systems, including notably some versions of RMPL. The Reactive Model-based Programming Language (RMPL) [105], another spinoff of the DS1/RAX experiment, proposes a common representation for planning, acting and monitoring. It combines a system model with a control model. The former uses hierarchical constraint-based automata to specify nominal as well as failure state transitions, together with their constraints. The latter uses reactive programming constructs (including primitives to address constraint-based monitoring, e.g., as in the Esterel [55] synchronous langage). RMPL programs are transformed into Temporal Plan Networks (TPN) [224], an extension of Simple Temporal Networks (STN) with symbolic constraints and decision nodes. Planning with a TPN is finding a path in the network that meets the constraints.

Altogether, RMPL integrates state-based models, procedural control and temporal reasoning. More recent work on RMPL focuses on the execution component of generated plans with choices and on the reduction of the plan size to improve the dynamic execution [51]. In [68], the authors extend TPN with error recovery, temporal flexibility, and conditional execution based on the state of the world. Each primitive task is given with a distribution of its likely duration. A probabilistic particle-sampling dynamic execution algorithm finds an execution guaranteed to succeed with a given probability. Probabilistic TPN are introduced in [192] with the notions of weak and strong consistency, as well as algorithm to check these properties. Finally, TPNUs [140] add the notion of uncertainty for contingent decisions taken by the environment or another agent. The acting system adapts the execution to observations and predictions based on the plan. It has been illustrated with a service robot which observes and assists a human.

RMPL is the most comprehensive CSP-based approach: it provides refinement, instantiation, time, nondeterminism, plan repair. The other systems handle very partially nondeterminism with local failure and conditional branching. Another important point is that all CSP approaches rely on common models to produce plans, skills and commands (often the separation between planning and acting is not sharp). These models can be verified and checked with formal tools [39, 2]. Yet, writing these models and debugging them remains tedious and not completely natural for the robot programmer. Finally, the performance of CSP approaches is not always up to real-time acting requirements, and some rely on another computational framework to implement the lower level components (e.g., [142]).

4.6. Acting with MDP-based approaches

As seen in Section 3.3, a policy π provides an acting procedure robust to nondeterminism and execution failures. It simply triggers the action given by π for the current state, observes the resulting state, and repeats until a goal is reached, or π is undefined. In POMDPs, acting is with respect to the belief state.

This is illustrated in the Pearl system of [171]. Although the domain in that case was fairly small (6 state variables giving a total of 516 states), the flat belief space was still beyond available algorithms. The authors underline that in their case, as in most applications, actions are only applicable in certain situations. A hierarchical partition of the action space (similar to HTN) leads to a tree whose leaves are small POMDPs. This hierarchical POMDP is tractable with near-optimal algorithms. The approach was tested for a robot assisting elderly individuals with mild cognitive and physical impairments and supporting nurses. The POMDP model is appealing in theory but quite restrictive in practice. Generally, many of the needed state variables in a robot can be made observable, only a few remain hidden. The robot's partial observability concerns only the latter. This significantly focuses the uncertainty about the current state and the possible courses of action. Such a partial observability approach is pursued with the MOMDP model [166, 8] which represents the states space as the cartesian product of a set of visible states and a set of hidden states.

Probabilistic approaches are also appealing for the interleaving of planning and acting. Algorithms such as RTDP [13] and LRTDP [26] offer an *anytime* approach interleaving planning and acting, when both functions are defined on the same state and action spaces. In urgency, the robot acts along a locally perceived best action. If it can afford a look ahead before acting, it explores part of its search space by performing as many random trials as it can afford to improve its value estimates. More elaborate sampling techniques offer some advantages:

- they allow the system to sample with predictive models and simulators as well as by acting in the environment,
- they naturally embed online receding horizon planning and acting, and
- they do not require a priori estimates of the model's probability distributions, as long as the samples are drawn from the same distribution.

Despite their advantages, there are relatively few robotic systems acting and controlling a plan with probabilistic models. This is probably due to the fact that the MDP-based approaches do not yet address the various types of processing needed for acting (action refinement, instantiation and constraint propagation, time management, etc.), in addition to decision making in nondeterministic domains.

For that, heterogeneous approaches mixing MDPs with other techniques offer a practical alternative. But the computational and functional properties of these approaches can be difficult to characterize. The Robel system [152] provides an example where planning and action refinement into skills (with HTN) are deterministic, but probabilistic techniques are used online, with receding horizon control, for choosing the best skill at each navigation step. It demonstrates that MDPs are well adapted for deciding about the appropriate skill into which an action is refined, in particular when that skill may change while the action is going on. A similar idea is illustrated in [239] for the control of the mission of a planetary rover. The input is a sequence of tasks representing the mission plan. Each task is a sequence of steps; a step is achieved by a module chosen in an explicit list. An MDP-based online system chooses the module for the next step and decide to continue the current task or to skip it, given available resources and expected rewards. These approaches require significant engineering for the design of their state space.

The Ressac project illustrates another MDP approach for achieving a UAV rescue mission [207]. Here the target area has been surveyed and segmented into a few zones where landing seems possible. An online anytime MDP planner interleaved with acting and control optimizes the final exploration, choice of a zone, and landing.

In conclusion, probabilistic techniques can be well adapted for refining and controlling a complex action. They need to be complemented with other approaches for controlling an entire plan. They require an expert modeler, knowledgeable in robotics and in MDP techniques, for structuring the state space and specifying the domain.

4.7. Summary

The acting function is in charge of refining planned actions into commands and reacting to events, while instantiating the remaining parameters in the plan, synchronizing and dispatching activities in a context dependent manner. Adequate integration between planning and acting is critical.

Early systems assumed that planning primitives were directly executable, reducing the acting function to reacting without much refinement deliberation. Procedure-based systems remain quite popular and widely developed in robotics systems. They offer advantages in flexibility, real-time properties and expressivity for specifying closed-loop skills. Their drawback is the demanding engineering effort required for the specification of skills. They raise significant obstacles for verification and validation, which are mandatory in safety critical applications.

Automata-based approaches address part of the latter concern. They offer well founded properties and tools for verification from hierarchical distributed specifications. The composition of automata significantly enrich deliberation capabilities. Future work will address time, uncertainty, nondeterminism and easier integration to planning. The Situation Calculus approach is used in a few systems. It offers a mixture of hand-specified procedures (as in logic-based programs) and reasoning capabilities. It is no as flexible as procedure-based systems and as rich in functionality. Further progress of formal Action Specification Languages in task planning may increase its usability for robotics.

Several of the analyzed systems rely on CSP-based techniques. They are designed for handling time, resources, uncertainty, and more generally when plans are expressed as consistent networks of constraints. Acting proceeds by making commitments and propagating additional constraints. The link to plan repair through CSP mechanisms is quite natural. Decision-theoretic approaches bring essential features for handling uncertainty, partial observability and dealing with closed-loop plans. They are not as widely used in robotics deliberation systems as one might expect,

in particular because of a more demanding model specification phase. Progress in learning methods will probably foster their development and use at the acting level of deliberation.

5. Monitoring and Goal Reasoning

Monitoring is a function in charge of *(i)* detecting discrepancies between predictions and observations, *(ii)* diagnosing their possible causes, and *(iii)* recovering from them. Monitoring is sometimes referred to as supervision or fault detection, identification and recovery (FDIR). More generally, monitoring has to detect if the robot's activity remains on course or if a course correction is needed.

At the highest level, a monitoring system has to survey whether the robot's current goals are still feasible and relevant for its mission, and possibly establish new goals if they are not (Sec. 5.4). While current goals remain valid, monitoring has to survey the planner's predictions supporting the plan achieving these goals. It also needs to survey predictions made when refining planned actions into commands, and to monitor conditions relevant for the current actions that are left implicit in the planning and refinement steps. The latter are, for example, the functioning status of the hardware components: the readiness conditions of the robot's sensors, or the charge level of its batteries. Finally, monitoring is closely linked to observing. Relevant information about the environment may be difficult or costly to sense and maintain continuously. An appropriate focus of attention should be set in order to detect conditions relevant for the current context and predictions.

As a function, monitoring is clearly distinct from acting. While the latter is a transformation process, the former has to keep a check on this process, including its motivations, which requires specific observation and anticipation. However, the overlap between acting and monitoring is often large in the literature and implementations. For example, the early Planex system performed very simple monitoring through the iterated computation of the current active kernel of a triangle table [73]. In most instances of procedure-based systems (Sec. 4.2), there are specific procedures in PRS, RAP, ACT or TCA, or part of such procedures, that handle monitoring functions. However, diagnosis and recovery functions in such systems are usually limited and quite ad hoc.

Monitoring in robotics takes often inspiration from the field of industrial control, where monitoring is a critical and well studied issue. Several monitoring systems, along that line, are surveyed in [169], most of which are mainly concerned with a robot sensory-motor level. These systems rely on control theory models, filtering techniques, statistical methods, such as Principle Component Analysis, for analyzing training data of normal and failures cases, and pattern recognition techniques for diagnosis. The relationship between monitoring and planning was not a major concern for the work discussed in [169]. This link (covered in Sec. 5.2 and 5.3) is essential from the integrated deliberation viewpoint of this survey. We'll also discuss relevant contributions to model-based diagnosis and to goal reasoning. Approaches presented below are summarized in Table 7.

	Knowledge Representation	**Search & Algorithm**	**Integration with other Deliberative Functions**
Planex [73]	Triangular table	Kernel method	PA
Livingstone [157]	Qualitative model Transition automata with probabilities and costs	ITMS with conflict-directed best-first search	PO
RMPL [105]	Hierarchical constraintbased automata	Reactive programming	P
Fraser [80]	Invariants	Satisfiability	P
Fichtner [72]	Fluent Calculus	Prioritized non-monotonic default logic	PO
Lamine [19]	Linear Temporal Logic	Delayed formula progression	P
Pettersson [170]	Neural net		PO
SKEMon [28]	Description Logic & Bayesian Belief		PO
TALplanner [63]	Temporal Action Logic	Formula progression	PO

Table 7: Surveyed monitoring approaches: knowledge representation, algorithms and links with other deliberation functions.

5.1. Model-based Diagnosis Approaches

Model-based approaches are exemplified by Livingstone [157, 20], the comprehensive spacecraft monitoring, diagnosis and recovery system in DS1/RAX. Livingstone relies on a knowledge representation specific to *qualitative*

model-based diagnosis[227]. Its objective is not to track the exact state of a system but simply to rule out incorrect diagnoses. The approach represents continuous functioning modes as nominal or qualitative deviations from nominal behaviors. The spacecraft is modeled as a fine grained collection of components, e.g., a thrust valve. Each component is described by a graph where nodes are the normal functioning states or the failure states of that component, e.g., valve closed, open or stuck. Edges are either nominal transition *commands* or exogenous transition *failures*. The latter are labeled by transition probabilities; the former are associated with transition costs and the preconditions of the command. A node is associated with a set of finite domain constraints describing the component's properties in that state, e.g., in the valve closed state, *inflow* = 0 and *outflow* = 0. The dynamics of each component is constrained such that, at any time, exactly one nominal transition is enabled but zero or more failure transitions are possible. Models of all components are compositionally assembled into a system where concurrent transitions compatible with the constraints and preconditions take place. The entire model is compiled into a temporal propositional logic formula which is queried through a specific solver (with a truth maintenance and a conflict-directed best-first search). Two query modes are used:*(i) diagnosis,* i.e., find the most likely transitions consistent with the observations, and *(ii) recovery*, i.e., find the least cost commands that restore the system into a nominal state. This monitoring system is well integrated with the acting system. It reports the spacecraft functioning state at the appropriate level of abstraction. It computes a focused sequence of recovery commands that meets additional constraints specified by the acting system.

This approach was demonstrated as very effective for a spacecraft. However, it is very specific to robots where reliability is a highly critical design issue addressed through a number of redundant components (to the sacrifice of cost) permitting complex diagnosis and allowing sophisticated recovery actions. Furthermore, the models are platform specific: a change in the hardware would involve significant development costs. Finally, the approach focuses the monitoring on the robot itself, not on the robot–environment interactions (whose possible "faults" are not modeled). It can be qualified as a robust *proprioceptive monitoring* approach. It is unclear how it could deal with environment discrepancies, e.g., a service robot failing to open a door. More research in that direction is required, e.g., along the framework of [10] which reformulates model-based diagnosis problems as planning problems.

5.2. Plan-Invariant Approaches

Several authors have synthesized from the specification of planning domain state-reachability conditions to speed-up planning [185, 102, 21]. These techniques can be used to extract *plan invariants*, i.e., conditions that have to hold during the entire execution of a plan. Going further, [80] proposes *extended planning problems*, where the usual planning specifications are extended by logical formula stating invariant conditions that have to hold during the entire execution of a plan. These extended invariants are used for monitoring the execution of a plan. They allow to detect infeasible actions or goals earlier than would done by checking preconditions; violated effects are detected even after actions have been successfully achieved. Extended invariants also allow a system to monitor the effects of exogenous events and other conditions not influenced by the robot. The paper shows that planning operators and extended invariants are two distinct issues that have to be modeled and specified distinctly. However, the important link of monitoring to observing remains somewhat shallow in this approach, e.g., the observation function assumes perfect and complete sensing. No empirical evaluation has been reported.

Along the same line, the approach of [72] has been tested on a simple office delivery robot. It relies on a representation of dynamic environment using the fluent calculus [191]. Actions are described by *normal* and *abnormal* preconditions. The former are the usual conditions, to be planned for before the actions. The latter are assumed away by the planner as default; they are used as a possible explanation of a failure. For example, delivery of an object to a person may fail with abnormal preconditions of the object being lost or the person not being traceable. Abnormal effects are similarly specified. Discrepancies between expectations and observations are handled by a prioritized non-monotonic default logic, and entail that default assumptions no longer hold. These explanations are ranked using relative likelihood, when available. The system is able to handle incomplete world models and observation updates received while acting or on demand from the monitoring system through specific sensory actions.

The idea of using extended logical specifications for planning and monitoring has been explored by several authors, in different settings. An interesting variant is illustrated in [19] for a hybrid robot architecture that combines behavior-based reactive control with model-based deliberation capabilities. At each cycle, concurrent behaviors are activated and combined (with some priority scheme) to determine the low level controls. At a higher level, properties of the robot behaviors are modeled using Linear Temporal Logic: LTL formulas are used to express goals as well as correctness statements and execution progress conditions. A trace of the robot execution, observed or predicted

at planning time, is incrementally checked for satisfied and violated LTL formulas. For that, a *delayed formula progression* technique evaluates at each state the set of pending formulas; it returns the set of formulas that has to be satisfied by any remaining trace. The same technique is used both for planning (with additional precondition-effect operators and some search mechanism) and for monitoring. The approach has been tested in indoor navigation tasks with Pioneer robots running the Saphira architecture [126].

At the low level of a behavior-based architecture, and complementary to the previous approach, the behavior activation signals can be tracked for patterns characteristic of the execution status. [170] developed a pattern recognition approach to exploit these activation signals. They train a neural net on various normal and faulty executions and use it online to detect and classify faults, reporting very low error rates in navigation tasks.

Finally, let us mention the interesting approach of [28] that uses domain knowledge expressed in description logic to derive expectations of the effects of actions in a plan to be monitored during execution. A first order query language allows online matching of these expectations against observations. Basically, the action parameters refer to world objects which have derived properties. These properties are checked to be either consistent or inconsistent with observations. They may even be of an undetermined consistency, permitting the monitoring function to deal with an open world semantics. An interesting extension handles flexible monitoring with probabilistic models, akin to Bayesien belief update. It relies on probabilistic plans with nondeterministic actions as well as on probabilistic sensing models. The approach was tested on indoor robot navigation and on simulated manipulation tasks.

5.3. Temporal Action Logic Approach

A comprehensive and coherent integration of planning and monitoring is illustrated in [63]. That system demonstrates a complex planning, acting and monitoring architecture embedded on autonomous UAVs. It has been tested in particular in surveillance and rescue missions. Planning relies on a forward chaining planner using the *Temporal Action Logics* (TAL) formalism for specifying operators and domain knowledge [133]. Formal specifications of global constraints and dependencies, together with planning operators and control rules, are used by the planner to control and prune the search. *Monitoring formulas* are generated from the model of planning operator (its preconditions, effects and temporal constraints), and from the complete synthesized plan, e.g., constraints on the persistence of causal links. This automated synthesis of monitoring formulas is not systematic but rather selective, on the basis of hand-programmed conditions of what needs to be monitored and what doesn't. Additional monitoring formulas are also specified along with the planning domain knowledge in the same highly expressive temporal logic formalism.

The TAL-based system produces plans with concurrent and durative actions together with conditions to be monitored during execution. These conditions are evaluated online using formula progression techniques. When actions do not achieve their desired results, or when some other conditions fail, recovery via a plan repair phase is triggered. Acting is performed by *Task Procedures* (TP) [62], which provide some level of action refinement through classical concurrent procedural execution, down to elementary commands. Observing uses *DyKnow* [101], a data-flow middleware to abstract and anchor sensor data (more in Sec. 6). Altogether, this system proposes a coherent continuum from planning to acting, observing and monitoring. The only component which does not rely on formal specifications is the acting function with hand written TPs. The lack of action refinement is addressed by specifying planning operators at a low level of granularity. For example, there are five different *fly* operators in the UAV domain, defining context-specific control and monitoring conditions and mapped to different Task Procedures.

5.4. Goal Reasoning

As introduced earlier, goal reasoning is a monitoring function at the highest level of mission management. Its role is to survey the current goals of a robot, check that they remain feasible and relevant for the mission at hand, and establish new goals if needed. It is sometime referred to as Goal Driven Autonomy (GDA). It is attracting a growing interest in AI, as revealed in the two surveys of [99, 213]. Although not focused on deliberation in robotics, these two papers are very relevant to our purpose. The former surveys a number of architectures supporting goal reasoning in intelligent systems (e.g., SOAR, BDI and other belief architectures). The latter reviews over 80 contributions on various techniques for goal monitoring, goal formulation and goal management, organized within a comprehensive goal reasoning analysis framework. We will review in this subsection a few examples of goal reasoning mechanisms that can be integrated to a robot's deliberation system.

Part of the goal reasoning function is sometimes embedded into the planner or the controller. However, goal reasoning does not produce plans; it manages current goals and establishes new ones which are passed to a planner.

As a monitoring function, it continuously checks for unexpected events or situations. It does not rely on the explicit prediction of the current plan (e.g., with invariants), and it may react to unexpected events by establishing new goals.

Goal Driven Autonomy (GDA) is generally presented as a reflexive model for reasoning about possibly conflicting goals and synthesizing new ones. In [150], the authors instantiate the GDA model in the ARTUE (Autonomous Response to Unexpected Events) system which appropriately responds to unexpected events in complex military simulations. ARTUE includes an HTN planner. If it detects a discrepancy when it executes a plan, it generates an explanation, possibly produces a new goal and manages possible conflict between goals currently under consideration. It uses different approaches to decide which goal to keep, e.g., decision theory. The approach proposes an original explanation system, which uses an Assumption-based Truth Maintenance System to find the possible worlds explaining the observed facts; it generates new goals, based on user-defined knowledge, to respond to the assumptions leading to these discrepancies. In [174], the authors extend ARTUE with a facility for teaching the system new goal selection rules through interactions with a teacher using active learning.In [230] the authors present an instance of GDA applied to AUV.

The Plan Management Agent (PMA) [173] provides, beyond planning and monitoring, the following functions:

- Commitment Management: commits to a plan already produced, and avoids as much as possible new plans which conflict with the existing ones. When a conflict arises, the agent reasons and decides which plan to give up.
- Alternative Assessment: decides which of the possible alternative goals and plans should be kept or discarded.
- Plan Elaboration: decides when and how to generate a plan.
- Coordination with Other Agents: takes into account others' commitments and the cost of decisions involving their plans.

PMA relies heavily on temporal and causal reasoning; it is able to plan with partial commitments that can be further refined later. It has been applied in personal calendar and workflow systems.

Goal reasoning has been deployed in a few real experiments. The Mission Manager (MM) in the DS1/RAX experiment [157, 20] is a notable illustration. From the expected state at the end of the current plan, MM considers the overall mission goals (from a list of nominal ones) and profile, and determines which goals should be satisfied in the next horizon and planning window (one to two weeks). The selected goals are passed to the planner, together with constraints which need to be satisfied at waypoints identified by the MM (e.g., the amount of energy left in the batteries should be above a threshold at the end of the planning phase). Planning produces a new plan to achieve these goals, which will be appended to the current one. Similarly, in the CPEF framework [159], mostly used in military deployment problems, the Plan Manager is in charge of the overall system; it controls the generation of plans, providing appropriate goal information to the planner.

5.5. Summary

Model-based approaches are effective for the proprioceptive monitoring of the internal state of a robot in safety critical applications . Techniques demonstrated in space can probably be extended to self-driving cars and similar applications (column *(vi)* in table 1). However, they are not sufficient for supporting robust deliberative behavior in complex changing environments. Monitoring requires a close link to and an extension of planning and acting domain knowledge in order to express conditions to be monitored while carrying out a plan. The state of the art illustrates a few systems implementing monitoring functions where the knowledge representation link is addressed at the most abstract planning level. More work is required for a consistent integration of monitoring and acting. The design of appropriate knowledge representations for planning and acting should integrate as early as possible the requirements of monitoring. Finally, a high degree of autonomy over long term missions requires a goal reasoning capability, a form of monitoring at the most abstract level. Few robotic systems have reached a level which requires extending monitoring with assessment and resolution of conflicting decisions.

6. Observing

Perception is a critical component in robotics, e.g., in feedback control loops, visual servoing [41], or Simultaneous Localization and Mapping (SLAM) [11, 151]. These metric and probabilistic techniques, possibly enriched with topological and semantic data, e.g., as in [131, 132, 125], are very effective for navigation and mapping tasks. But robots face other chores beyond mobility.

The observing function provides the capability to identify objects, states and sequences of events relevant to the robot's activity. It has to be tightly integrated with planning, acting and monitoring. It combines bottom-up processes, from sensors to meaningful data, with top-down mechanisms such as focus of attention, reasoning with sensor models and planning with sensing actions. Observing functions of the systems presented below are summarized in Table 8.

	States and objects	**Situations**	**Planning**	**Knowledge Representation**
Coradeschi [53]	Anchoring	Anchor tracking	Link to planning	Anchors
PTL [115]	Ambigous anchors		Conditional planning	Anchors
IxTeT-Reco [65]		Chronicle recog.	Link to IxTeT Planner	Chronicles
SAM [167]		Chronicle recog.		Timelines
DyKnow [101]	Anchoring	Chronicle recog.	Integration with TALPlan	Streams, KPL
HiPPo [203]	Objet recognition		Hierarchical POMDP	POMDP
Velez [214]	Object recognition		Bayesian	DBN

Table 8: Surveyed observing systems: approaches and main functions.

6.1. Planning to perceive

We already discussed task planning with sensing and information gathering actions (Sec. 3.4). Planning to perceive is a distinct issue concerned with integrating the selection of viewpoints and sensor modalities with recognition, navigation and other tasks. It relies on extensive work on the sensor placement problem, usually addressed as a search for the next best viewpoint in a recognition or a modeling task, as in [135]. Several authors address this integrated planning problem with POMDPs, e.g., [171, 175]. The HiPPo system [203] offers a good illustration of sensor placement for the recognition of objects on a table, as typically required in a manipulation task where intractability issues are addressed with a hierarchical POMDP technique. The approach seems feasible in limited perception tasks, involving a few regions of interest.

An alternative and more scalable approach for synthesizing an observation plan within a navigation task is proposed in [214]. This work seeks to detect and map objects of interest while reaching a destination. It uses a Bayesian approach which correlates measurements from subsequent observations to improve object detection; detours are weighed against motion cost to produce robust observation plans using a receding horizon sampling scheme. The approach was tested in an indoor environment for recognizing doors and windows.

6.2. Object Recognition and Symbol Anchoring

The *anchoring problem* provides an excellent illustration of the complexity of integrating pattern recognition methods with deliberate action. As defined in [53], anchoring is the problem of creating and maintaining over time a correspondence between symbols and sensor data that refer to the same *physical object*. Planning and other deliberation functions reason about objects in the environment through symbolic attributes and relations linking these symbols. Observing handles perceptual data. It is essential that the abstract description of the former and the data of the latter agree. Anchoring concerns specific physical objects. It can be seen as a particular case of the *symbol grounding* problem, which deals with broad categories, e.g., any "chair", as opposed to that particular chair-2.

Anchoring an object of interest can be achieved by establishing and keeping a link, called an *anchor*, between the perceptual system and the symbol system, together with a signature that estimates some of the attributes of the object it refers to (see Fig. 6, taken from [54]). The anchor is based on a model that relates relations and attributes to perceptual features and their values. Establishing an anchor corresponds to a pattern recognition problem, with the usual challenge of handling uncertainty in sensor data and ambiguity in models, a challenge dealt with for example by maintaining multiple hypotheses. In [115], ambiguous anchors are handled with a conditional planner, called PTL, exploring a space of belief states, representing the incomplete and uncertain knowledge due to partial matching between symbolic properties and observed perceptual features. The authors distinguish between definite symbolic descriptions, which are matched with a single object, and indefinite descriptions. Actions have causal effects that change object properties. Observations can change the partition of a belief state into several new hypotheses. They experimented with an indoor robot equipped with vision and smell sensing in object recognition tasks.

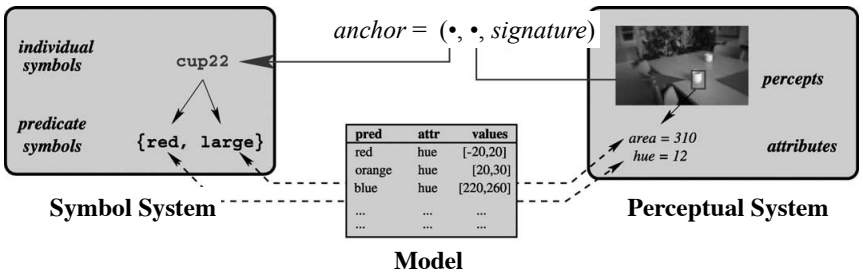


Figure 6: Anchor relating symbols and percepts [54].

Anchoring raises several additional problems, e.g., which anchors to establish, when and how. Anchors are needed in principle for all objects relevant to the robot mission. These objects can only be defined by intension (not extensionally), in a context-dependent way. Tracking anchors is another issue, *i.e.*, taking into account object properties that persist across time or evolve in a predictable way. Predictions are used to check that new observations are consistent with the anchor and that the updated anchor still satisfies the object's properties. Finally reacquiring an anchor when an object is re-observed after some time is a mixture of finding and tracking; if the object moves it can be quite complex to account consistently for its behavior (e.g., see the DyKnow system below).

Interaction with humans, in particular in natural language as in [196], or between robots, adds other problems of sharing anchors between distinct agents. For example [118] considers several robots cooperating in office navigation and object finding tasks by exchanging information about specific features (color, position) with shared anchors. There is also the issue of learning the models needed for building up anchors, illustrated in [30] through a supervised learning approach. For that, one may draw from the wealth of pattern recognition techniques in machine learning [23], and from the growing set of labeled data and corresponding models on the web.

6.3. Event and Situation Recognition

The dynamics of the environment are an essential source of information in the observing function, as well as a source of complexity, e.g., as we just saw in the anchor tracking and re-acquiring problems. These dynamics are themselves what needs to be interpreted: what an observed sequence of changes means, what can be predicted next from past evolutions. Research on these topics relates to acting in and understanding environments with rich semantics, in particular involving human and man–robot interactions, e.g., in applications (column *(iii)* in Table 1) such as smart human–support environments [167], robot programming by demonstration [9] or video surveillance [103, 82].

The survey of [129] covers an extensive list of contributions to action and plan recognition. These deal with *(i)* human action recognition, *(ii)* general activity recognition, and *(iii)* plan recognition. The former two types of processing provide input to the latter. Most surveyed approaches draw from two sources of techniques: signal processing and plan recognition. The former use filtering approaches, Markov Chains, and Hidden Markov Models (HMM, e.g., [179]). They have been successfully applied to movement tracking and gesture recognition [232, 149]. The latter rely on deterministic [116, 182] or probabilistic [87] planning techniques, as well as parsing techniques [177].

Most plan recognition approaches assume as input a sequence of symbolic actions. This assumption is hard to meet in robotics. Usually actions are sensed only through their effects on the environment. The recognition of actions from their effects depends strongly on the plan level. Decomposing into recognizing actions then plans from these actions is fragile. Hence, classical plan recognition techniques has a limited usefulness in robotics.

Chronicle recognition techniques [65, 88] are very relevant to the observing function. A chronicle is a model for a collection of possible scenarios. It describes classes of events, persistence assertions, non occurrence assertions and temporal constraints. A ground instance of a chronicle can be formalized as a nondeterministic timed automata. Chronicles are similar to IxTeT planning operators (Sec. 3.2). IxTeT-Reco, an extension of the IxTeT planner with the same representation, implements a chronicle recognition system. It is able to monitor a stream of observed events and recognize, on the fly, instances of modeled chronicles that match this stream. The recognition is efficiently performed by maintaining incrementally a hypothesis tree for each partially recognized chronicle instance. These trees are

updated or pruned as new events are observed or time advances. It has been demonstrated in robotics surveillance tasks. Recent development have introduced hierarchy and focus on rare events [66].

The chronicle approach is appealing in particular for its direct link to planning and observing. The SAM system [167] is a good illustration of the interest of such a link in the development of a ubiquitous robotics environment providing assistance to an elderly person. It uses a chronicle-like representation (timelines with interval algebra) offering online recognition, planning and execution with multiple hypotheses tracking over weeks.

6.4. Stream-Based Integration

Very few systems have been proposed for designing and implementing an integrated observing function, from signals, to states and chronicles. DyKnow [101] stands as a clear exception, noteworthy by its comprehensive and coherent approach. This system addresses most observing requirements: the integration of different sources of information, of hybrid symbolic and numeric data at different levels of abstraction with bottom-up and top-down processing, managing uncertainty, dynamically reconfigurable, and reasoning on explicit models of its content.

These requirements are addressed as a data-flow based publish-and-subscribe middleware architecture. DyKnow views the environment as consisting of objects described by a collection of *features*. A *stream* is a set of samples representing observations or estimations of the value of a feature. A sample has two time-stamps: when it was acquired and when the corresponding value was made available. A stream is associated with a formally specified *policy* giving requirements on its content. Examples of such requirements are the frequency of updates, the delays and amplitude differences between two successive samples, or how to handle missing values.

A stream is generated by a *process* which may offer several stream generators synthesizing streams according to specific policies. Processes have streams as input and output; they are of various types, e.g., *primitive processes* directly connected to sensors and geographic information systems; *refinement processes* which subscribe to input streams and provide as output combined features, e.g., a signal filter process or a position estimator process fusing several raw sensing sources and filtered data; *configuration processes* which allow reconfiguring dynamically the system by initiating and removing processes and streams as required by the task and the context, e.g., to track a newly detected target.

DyKnow uses a specific language, called KPL (Knowledge Processing Language), to specify processes, streams and corresponding policies. KPL allows in particular referring to objects, features, streams, processes, and time, together with their domains, constraints and relationships in the processing network. Formal specifications in KPL define a symbol for each computational unit, but they do not define the actual function associated with this symbol. Their semantics is taken with respect to the interpretation of the processing functions used. They allow the programmer to describe and enforce stream policies. They also allow the support several functions such as:

- synchronize states from separate unsynchronized streams;
- evaluate incrementally metric temporal logic formulas over states;
- recognize objects and build up anchors to incrementally update interpretation as new information becomes available;
- follow histories of spatio-temporal events and recognize occurrences of specified chronicle models.

Initially, DyKnow was implemented as a Corba middleware; the latest implementation relies on ROS middleware and topics. It has been integrated as the observing function of the TALplan system [63] discussed in the previous section. This system is naturally queried by planning, acting and monitoring functions to acquire information about the current contextual state of the world. It provides appropriate and highly valuable focus of attention mechanisms, linking monitoring or control formulas to streams. It has been deployed within complex UAV rescue and traffic surveillance demonstration (the latter is illustrated in Fig. 7, taken from [101]).

6.5. Summary

Robotics research in perception is mostly focused on mobility, localization and mapping. These are essential but not sufficient functions. As discussed earlier (Sec. 4.6), most systems are partially observable in the sense that some of their state variables are observable while a few are hidden. The former demand explicit sensing and processing actions to be acquired, tracked and adequately interpreted. Observing is a critical deliberation function.

We reviewed here contributions to perception planning, anchoring symbols and signals, and recognizing situations. We also discussed a comprehensive stream-based approach for integrating an observing system with planning, acting

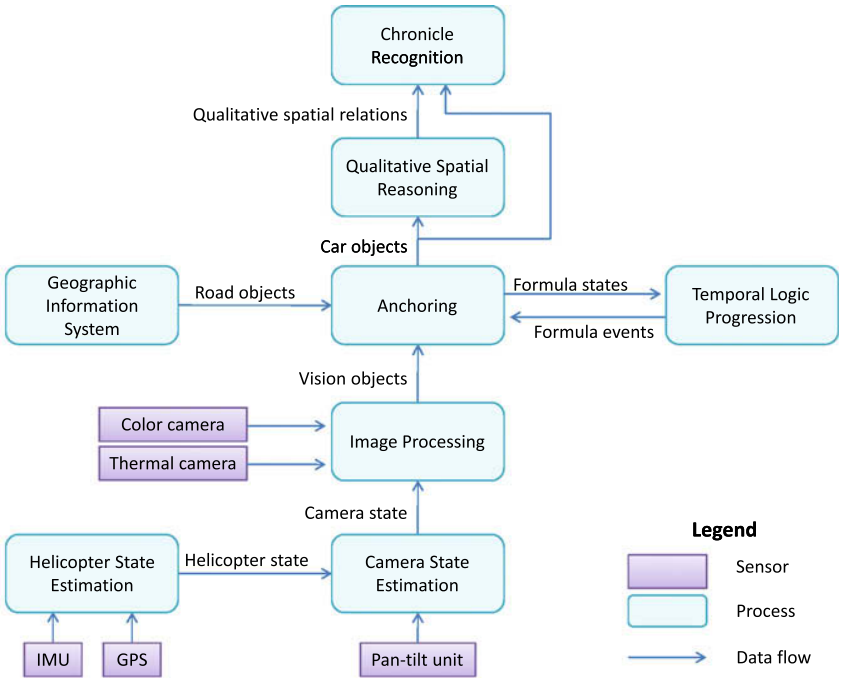


Figure 7: Dyknow processes for a UAV traffic surveillance task [101].

and monitoring. More work on similar systems is required to better qualify their capabilities and limitations for various robotics environments (in particular those marked "*h*" in column *(v)* and *(viii)* of Table 1).

7. Acquiring Deliberation Models

Deliberation requires a variety of models of the environment, of the robot's sensors, actuators, skills, tasks, etc. These models are often hand-specified in different representations. They can be tuned through estimation, parameter acquisition and calibration procedures. In a few cases, they can be verified and validated, usually at the level of one function, rarely when composed within a global behavior.

Modeling is a well-known bottleneck, more severe when the diversity of environments and tasks increases, i.e., when deliberation is needed. It is desirable to endow a robot with adaptation capability to allow it to accommodate to changes in its environment and tasks, to remove errors and improve its models and behaviors. Learning in robotics is a very active field. However, most of the effort has been devoted to two areas:

- *Sensory-motor control models*, as in [47, 121, 120, 164], and most of the work surveyed in [197];
- *Environment models*, in particular at the geometric level with SLAM techniques [209, 11], possibly combined with planning [145], and at the combined geometric and topologic levels, e.g., [45, 125, 184].

Learning sensory-motor and environment models is critical for autonomous robots, but it is out of the scope of this survey. Let us focus here on a few contributions to the acquisition of models needed for and specific to deliberation.

7.1. Acquiring semantic maps

Semantic maps extend geometric and topological models with semantic information about the environment. Contributions such as [84, 183] illustrate approaches for acquiring, organizing hierarchically, and optimizing such a map. Elvira [84] represents the environment as a hierarchy of graphs. Nodes at a given level are symbols referring to locations and places; arcs correspond to relations such as navigability or proximity. At the lowest level, anchoring

techniques capture the dynamics of the environment. Elvira contains an anchoring module, a hierarchy optimizer, and a hierarchical task planner. It has been tested on a hybrid behavior-based robot in navigation tasks.

Semantic maps endow a robot with deduction abilities about its environment that complement and enhance its perceiving function as well as its planning and acting functions. They allow the robot to relate a hierarchy of concepts and relations to a hierarchy of objects and places. For example, the concept "kitchen" is described as a topological node that contains objects such as stove, sink, etc. These objects are characterized with features permitting recognizing and using them. Techniques such as those mentioned in Section 5.2 are used to derive information about partially observed current states as well as expectations about the effects of actions. The Astrid indoor robot demonstrates in [83] how semantics maps are learned, how semantic knowledge is integrated with metric and topologic maps, and how it is used to plan and execute domestic tasks.

7.2. Acquiring planning and acting models

Numerous methods have been proposed for acquiring and improving action models at the skill and task levels.

Approaches for acquiring and improving skill models are for example [78] with HMM, or [104] with Dynamic Bayes Nets (DBN) [56]. Skill selection can be addressed as acquiring an MDP policy, as illustrated in the Robel system [152]. These three systems have been demonstrated in indoor navigation tasks. Similarly, the MRTE approach of [148] considers several skills for a given task; it uses a supervised learning approach where a teacher provides corrections for choosing a skill and commands within skills. It has been demonstrated for humanoid obstacle avoidance.

Learning in task planning has been used to speed-up planners with control knowledge and macro actions [162, 49], specific heuristics [236, 233] and their choice rules [64]. It has also been used to improve the quality of plans with respect to cost, success rate [143, 204], or user's preferences [6], and to learn domain models and planning operators [235, 220]. The survey of [240] analyses a wide spectrum of approaches ranging from decision trees, inductive logic, and explanation-based learning, to classification methods, Bayesien learning and neural nets. The more recent review of [109] updates this map with respect to new contributions for acquiring planning domain models or control knowledge. Most of these contributions are concerned with speeding up task planners; only a few consider learning while acting; even fewer are demonstrated in robotics. This is not the case for approaches in reinforcement learning and in learning from demonstration.

Reinforcement learning (RL) integrates learning and acting. Most approaches rely on learning an MDP policy, progressively improved with respect to a reward criterion [112, 205, 93]. RL approaches are usually non-supervised: the robot gets as feedback the cumulative rewards of its actions. RL raises the issue of *exploration* vs *exploitation*, i.e., exploring all possibilities to acquire new knowledge *vs* making use of learned knowledge [57]. Scaling RL methods with discrete state spaces is a major concern. Continuous state variables with parametric models allow overcoming this issue by using functional approximation and policy search methods [121]. They also allow generalizing what has been learned to unexplored areas of the parameter space when some continuity assumptions are met [189]. An illustration is given in [128] for learning grasping tasks by exploring the continuous parameter space with an approximation of the upper confidence bound on the return values using Gaussian process regression.

Generalization is also pursued through *policy reuse* techniques. These extend the exploration–exploitation alternative with a third option, the reuse of previously known or learned policies. Policy reuse is illustrated in RoboCup domains in [70]. The approach requires mappings between state and actions spaces, but no a priori information about when the reuse is useful.

With more diverse environments and tasks, flat propositional representations are limited in expressiveness and complexity; their convergence requires a huge amount of training. A few structured representations have been explored, among which: *factored MDPs* with compact DBN-like representations [219], *hierarchical RL* methods [60, 144], and *relational RL* methods [67]. These approaches should be more adapted for learning complex decomposable behaviors and integrating learning into the reasoning methods used in task planning. Their effective development in complex robotics tasks remains to be demonstrated.

RL methods face another issue: reward functions are difficult to specify and estimate from observable variables. Inverse reinforcement learning aims to acquire reward functions from demonstrated behavior [163, 1]. With a similar motivation, other approaches augment RL methods with feedback from and interaction with a teacher to get advice on preferred policies. In this supervised RL mode, a teacher discriminates (e.g., through preferences) between alternative trajectories of *(state, action)* pairs issued from the same state [229]. The approach of [111] allows the learner to choose

most useful states where advice from the teacher can be useful. It has been tested in several simulation benchmarks such as learning to balance a cart pole or to ride a bicycle. These interactive approaches can be considered as a form of learning from demonstration.

Learning from demonstration is a paradigm for various techniques allowing a robot to learn policies from behaviors demonstrated by a human tutor. The survey of [9] classifies approaches into *teleoperation, shadowing, instrumented teacher* and *external observation*. In the former two categories, demonstrations take place in the robot configuration space. In the latter two, demonstrations are in the natural space of the teacher. Sensing of the state variables is direct in the cases of teleoperation or instrumented teacher. It requires interpretation in the two other cases. Learning methods acquire either policies, state–action dynamics from which policies can be derived, or explicit action models permitting deliberation and planning. Teleoperation approaches have been widely experimented with, e.g., in kicking and grasping motions, assembly, avoidance, navigation or helicopter flying. Shadowing has been used in navigation (following a teacher), or for learning hand gestures. Instrumented teachers have taught various skills and cup-and-ball games. Very few techniques cover the last category with external observation, where clearly more research is needed. The recent book of [42] proposes future directions of research.

7.3. Summary

The state of the art for the acquisition of deliberation models remains very preliminary. We discussed a few approaches for acquiring semantic maps and integrating the learned knowledge into planning and acting models. These and similar approaches are strongly needed in service and domestic robots and other applications with similar features (see Table 1). We also discussed a few Reinforcement learning and Learning from Demonstration approaches which appear as promising research direction that have not yet been developed at the deliberation level in robotics.

8. Discussion and Synthesis

Numerous deliberation systems were discussed in Sections 3 through 7, along their planning, acting, monitoring, observing and learning functions. A synthesis of their common features, their significant differences, weak points and open challenges is in order. We will focus this synthesis on the analyzed systems in Table 2.

All the analyzed systems are concerned with three classes of applications: service and domestic robots, exploration and rescue robots (aerial, underwater and ground) and spacecraft. This is explained by the features of applications in Table 1 and from the fact that deliberation is entailed by autonomy and diversity of tasks and environment. It is also due to the state of the art which made it more feasible and robust to engineer out the environment in manufacturing and industrial robotics, even if this situation is now changing. Further development will make deliberation systems more effective in functionality and cost for most robotics applications.

Most analyzed systems address three main functions: planning, acting and monitoring. It is interesting to note that, in most of these systems, planning explicitly takes into account space and time. The former is mandatory for a mobile robot, but of particular cases, e.g., a spacecraft. There is however a significant difference between embedding a motion planner in a system's architecture, as a GenoM module in [3] or in the WITAS system [61], and integrating motion and task features in the search and state spaces of planning, as done in Asymov [36] and other systems discussed in Section 3.1. The latter option can properly handle interdependences between space configurations and other task properties. With the growing interest in the tight coupling of motion and task planning, one can expect to see their integration more prevalent in robotic deliberation systems.

Most planners in the analyzed systems handle time and concurrency. These are indeed important features which allows for simultaneous activities and synchronization with events in dynamic environments. Some systems deploy a temporal planning approach (e.g., IxTeT, RAX-PS, EUROPA, Aspen, TALPlanner) with state variables over timelines, while others have a narrower management of time as a resource (e.g., Cypress), or as a timed behavior extension to Hierarchical Constraint Automata for RMPL, and timeline management in XFRM. The simpler and more efficient state space task planning approaches are very relevant in many applications; it is unclear whether they can replace temporal planning altogether. Probabilistic and other nondeterministic planners have, for the moment, a narrower use, mostly at the sensory-motor level, e.g., within reinforcement learning approaches. Their integration in deliberation system will certainly be developed, in particular at the acting level. Finally, the predictive capability of the models in most of the surveyed planners remains quite narrow and limited. One should expect significant progress from simulation-based planning, e.g., with sampling search and receding horizon techniques.

A few of the analyzed systems consider planning primitives as executable commands; this limits the acting function mostly to reacting to events.

Most systems do, however, introduce a refinement level between planning primitives and commands. The majority use different knowledge representations for planning and acting. The exceptions are: RMPL, which relies on one model (hierarchical constraint automata with reactive programs) but provides limited planning capabilities; IDEA and T-REX, which proposes a single agent/reactor representation using state variables over timelines; and Cypress, which relies on the ACT interlingua as a rich programming environment for specifying both HTN planning domains and PRS acting procedures. However, the integration in ACT of planning and acting remains quite shallow, e.g., there is a de facto partition between planning methods and acting procedures.

This issue of an integrated representation between planning and acting is clearly a weak point in analyzed systems (except for IDEA and T-REX with their state variables hierarchy and the continuum between acting and planning). Such a representation should clarify the various refinement levels (at least two, possibly more) in terms of different state spaces and actions spaces and the relation between these spaces, e.g., the state space at a level i being a refinement of the more abstract level $i-1$, with a clear mapping from i to $i-1$ spaces.

As seen in Section 3, abstraction in planning has been extensively studied, mostly to reduce complexity; it remains to be extended at the architecture level and for different forms of plans and skills. This is needed for specification, verification of consistency, and integration of planning and acting knowledge.

Monitoring is the third function common to most analyzed systems. The link between planning and monitoring knowledge is well advanced. In a few cases, both rely on the same knowledge representation, e.g., Temporal Action Logic in the WITAS/TALPlan approach or Temporal Plan Networks in the RMPL system. The notion of monitoring formulas synthesized from plan invariants and from additional domain knowledge seems quite effective and should be more widely deployed in deliberation systems.

The integration of an observing function in the deliberation system is less frequent in the analyzed approaches. It is quite narrow in DS1/RAX and IxTeT-Reco (despite the common chronicle representation for planning and observing). It is more comprehensive in DyKnow. The interesting perception planning facilities in PTL, HiPPo and other systems discussed in section 6 and their integration to navigation and task planning have to be more widely developed and incorporated into deliberation systems.

A few of the analyzed systems offer some limited form of goal reasoning, notably DS1/RAX and Cypress. The incentives in terms of autonomy at the mission level were possibly less strong than they are today. Techniques such as those of ARTUE and others discussed in section 5.4 are certainly relevant in robotics and should be integrated in many deliberation systems.

Learning appears in only one of the analyzed systems: XFRM/RPL/CRAM. Our discussion in Section 7 indicates that learning is quite successful in robotics at the sensory-motor level. There will be significant progress in deliberation if more advances in different forms of reinforcement learning and learning from demonstration take place at the planning and acting levels. Meanwhile the techniques of Elvira [83] for acquiring hierarchical semantic maps and using them in planning should be more widely used in rich environment robotics applications.

This last point is related to what we called Open Architectures and planning in open domains (Sec. 3.4), illustrated in the CRAM system, and, for some aspects, in WITAS (link to a Geographical Information System). Such an opening may not seem to be a radical change in representation and reasoning capabilities. It offers, however, a huge set of opportunities for adapting and extending deliberation models and dealing with complex environments. Most robotics deliberation systems should probably integrate similar capabilities.

9. Conclusion and Perspectives

Deliberate action is a central research issue in robotics. It is an issue motivated by critical capabilities needed by autonomous robots facing a variety of environments, tasks and interactions. Many applications in exploration, service, and personal robotics depend greatly on further progress and development of robotic deliberation capabilities.

Numerous contributions for the design of deliberation functions for a robot have been proposed. A few have been discussed in the preceding sections. The state of the art is broad and rich, but quite fragmented. The relationships between these fragments are not studied enough. It is indeed clear from our survey that only a few approaches take a global integrative view. One finds more results about how to speedup a given technique then comprehensive characterizations of that technique, how it complements other approaches, and how it can be integrated with them.

For the purpose of this survey, we have outlined six deliberation functions and discussed their requirements and characteristics. These functions may widely overlap in a given implementation. The rational for their organization within an operational architecture has to take into account numerous requirements, in particular a hierarchy of closed loops, from the most dynamic inner loop, closest to the sensory-motor functions, to the most external outer loop.

A critical aspect is the relationship between planning and acting. Acting cannot be reduced to the reactive triggering of sensory-motor commands mapped from planned actions or observed events. There is a need for significant deliberation between what is planned and the commands achieving it. This deliberation may even use different planning techniques, in particular motion, manipulation and other domain specific techniques. It has to be done in different yet well integrated state spaces, action spaces and event spaces. Although we insisted on distinguishing the two functions of acting and planning on top of the execution platform, there is no reason to believe that just three levels is the right number. There can be a hierarchy of planning–acting levels, each refining a task planned further up into more concrete actions, adapted to the acting context and foreseen events. It would be elegant to address this hierarchy within a homogeneous approach. However, conflicting requirements, e.g., for handling domain specific representations and different levels of nondeterminism, uncertainty and inaccuracy, favor a variety of representations and approaches.

Deliberation functions encompass other open issues that we have not discussed in this survey. Notable among them are the problems of *interaction* and *social behavior*. Interaction impacts all functions discussed here, from the observing requirements in a multi-modal dialogue, to planning and acting at the levels of task sharing and plan understanding for multi-robots and human-robot interaction.

In summary, the take home message of this survey is that:

- deliberation in robotics does not reduce to task planning,
- acting is a context and task dependent reaction to events and refinement of planned actions into commands,
- hierarchy of action refinements over several levels of state and action spaces is essential,
- monitoring needs strong links to prediction in planning and acting,
- observing requires significant deliberation, it does not reduce to signal processing,
- integrated and consistent representations and the acquisition of corresponding models are a clear bottleneck.

Future research directions emphasize representation, modeling and integration issues at various levels:

- *Descriptive models* for task planning tend to favor the performance of search algorithms at the cost of shallow predictions. Often in robotics, available knowledge, changing environments and the type of missions do not allow for or require plans with thousands of actions, at a proper level of abstraction. The bottleneck is more on improving prediction capabilities of the models then on scaling up of search techniques. Promising directions include detailed simulation systems and sampling techniques. Heterogeneous knowledge representations can be desirable, although more difficult to integrate and verify.
- *Operational models* for acting, to refine planned actions and react to events, have to lead to executable procedures, but they should not be just executable code. They should allow for several decision making functions, e.g., choice, instantiation, look-ahead, time and context handling, etc. Here also predictive capabilities are needed, but at a shorter range and in a closed loop with perception.
- *Open environment models* and their relationship with descriptive and operational models for planning and acting offer exciting problems and perspectives. Robots do not need to reinvent the wheel. Their deliberation capabilities may build on communication and social mechanisms.
- *Model acquisition* is certainly a bottleneck for deliberation functions. Most approaches surveyed in this paper demand strong robotics expertise for designing and structuring required models. *Machine learning* techniques are critical for the development of deliberative robots; they lead to numerous open problems. Learning should be combined with model synthesis from higher level specifications. For example, monitoring models should be synthesized from the operational and descriptive models for planning and acting.
- *Model verification* problems have also to be stressed, e.g., in order to relate operational models to descriptive models, hand-specified or learned. Consistency verification is harder with heterogeneous representations. In many cases, verification can be a powerful tool in model acquisition techniques.
- The observing function and its integration with other deliberation functions involve very challenging open problems in deliberation robotics, covering a wide spectrum from anchoring to symbol grounding, from object recognition to functional categorization, and from semantic map building to situation and dynamic environment understanding.

- Finally, *integration and architecture* issues remain critical in the design of a deliberation system. The conceptual view of a few deliberation functions sharing knowledge and data and interacting with sensory-motor functions serves a clarifying purpose. It needs to be mapped into a more effective architecture whose organizational principle is a hierarchy of deliberation and sensory-motor functions relying on a hierarchy of clearly related representations and models, allowing for a plasticity of reconfiguration, as well as for sharing models and developments among roboticists as well as between robots.

Acknowledgments

We are very grateful to the anonymous reviewers and to the editors of this special issue for their thorough reading and insightful comments, which contributed to improving this paper. Many thanks to several colleagues, in particular Wheeler Ruml and Frédéric Py for their very helpful technical and editorial feedback. This work was supported in part by EU FP7 SAPHARI under grant agreement No. ICT-287513.

References

[1] Abbeel, P., Ng, A. Y., 2010. Inverse Reinforcement Learning. In: Sammut, C., Webb, G. I. (Eds.), Encyclopedia of Machine Learning. Springer, pp. 554–558.
[2] Abdeddaïm, Y., Asarin, E., Gallien, M., Ingrand, F., Lesire, C., Sighireanu, M., 2007. Planning Robust Temporal Plans: A Comparison Between CBTP and TGA Approaches. In: ICAPS.
[3] Alami, R., Chatilla, R., Fleury, S., Ghallab, M., Ingrand, F., 1998. An Architecture for Autonomy. IJRR 17 (4), 315–337.
[4] Alcázar, V., veloso, M. M., Borrajo, D., 2011. Adapting a Rapidly-Exploring Random Tree for Automated Planning. In: SoCS.
[5] Allen, J. F., Jul. 1984. Towards a General Theory of Action and Time. Artificial Intelligence 23 (2), 123–154.
[6] Ambite, J. L., Knoblock, C. A., Minton, S., 2000. Learning Plan Rewriting Rules. In: ICAPS.
[7] Ambros-Ingerson, J., Steel, S., 1988. Integrating Planning, Execution and Monitoring. In: AAAI.
[8] Araya-Lopez, M., Thomas, V., Buffet, O., Charpillet, F., 2010. A Closer Look at MOMDPs. In: ICTAI.
[9] Argall, B. D., Chernova, S., veloso, M. M., Browning, B., 2009. A survey of robot learning from demonstration. Robotics and Autonomous Systems 57 (5), 469–483.
[10] Baier, J. A., Mombourquette, B., McIlraith, S., 2014. Diagnostic Problem Solving: A Planning Perspective. In: KR. pp. 1–10.
[11] Bailey, T., Durrant-Whyte, H., 2006. Simultaneous localization and mapping (SLAM): part II. IEEE RAM 13 (3), 108–117.
[12] Barbier, M., Gabard, J.-F., Llareus, J. H., Tessier, C., 2006. Implementation and Flight testing of an onboard architecture for mission supervision. In: Bristol International Unmanned Air Vehicle Systems Conference.
[13] Barto, A. G., Bradtke, S. J., Singh, S. P., 1995. Learning to Act Using Real-Time Dynamic Programming. Artificial Intelligence 72 (1-2), 81–138.
[14] Beetz, M., 1999. Structured reactive controllers: controlling robots that perform everyday activity. In: AGENTS. ACM, pp. 228–235.
[15] Beetz, M., McDermott, D., 1992. Declarative goals in reactive plans. In: AIPS.
[16] Beetz, M., McDermott, D., 1994. Improving Robot Plans During Their Execution. In: AIPS.
[17] Beetz, M., McDermott, D., 1997. Expressing transformations of structured reactive plans. In: ECP.
[18] Beetz, M., Mösenlechner, L., Tenorth, M., 2010. CRAM—A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In: IROS.
[19] Ben Lamine, K., Kabanza, F., 2002. Reasoning about Robot Actions: A Model Checking Approach. In: Beetz, M., Hertzberg, J., Ghallab, M., Pollack, M. E. (Eds.), Advances in Plan-Based Control of Robotic Agents. Springer, pp. 123–139.
[20] Bernard, D., Gamble, E., Rouquette, N., Smith, B., Tung, Y., Muscettola, N., Dorais, G., Kanefsky, B., Kurien, J. A., Millar, W., 2000. Remote agent experiment ds1 technology validation report. Tech. rep., NASA.
[21] Bernardini, S., Smith, D. E., 2011. Finding Mutual Exclusion Invariants in Temporal Planning Domains. In: IWPSS.
[22] Bidot, J., Karlsson, L., Lagriffoul, F., Saffiotti, A., 2014. Geometric Backtracking for Combined Task and Path Planning in Robotic Systems. Tech. Rep. 1, Applied Autonomous Sensor Systems, Örebro University, Sweden.
[23] Bishop, C. M., 2006. Pattern Recognition and Machine Learning. Springer.
[24] Bohren, J., Rusu, R. B., Jones, E. G., Marder-Eppstein, E., Pantofaru, C., Wise, M., Mösenlechner, L., Meeussen, W., Holzer, S., 2011. Towards autonomous robotic butlers: Lessons learned with the PR2. In: ICRA. pp. 5568–5575.
[25] Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P., Slack, M., Apr. 1997. Experiences with an Architecture for Intelligent, Reactive Agents. JETAI 9 (2/3), 237–256.
[26] Bonet, B., Geffner, H., 2006. Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and Its Application to MDPs. In: ICAPS.
[27] Bonet, B., Geffner, H., 2006. Learning in Depth-First Search: A Unified Approach to Heuristic Search in Deterministic, Non-Deterministic, Probabilistic, and Game Tree Settings. In: ICAPS.
[28] Bouguerra, A., Karlsson, L., Saffiotti, A., 2007. Semantic Knowledge-Based Execution Monitoring for Mobile Robots. In: ICRA. pp. 3693–3698.
[29] Boutilier, C., Dean, T., Hanks, S., May 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. JAIR 11, 1–94.

[30] Bredeche, N., Chevaleyre, Y., Zucker, J.-D., Drogoul, A., Sabah, G., 2003. A meta-learning approach to ground symbols from visual percepts. Robotics and Autonomous Systems 43, 149–162.
[31] Brenner, M., Nebel, B., Jun. 2009. Continual planning and acting in dynamic multiagent environments. Autonomous Agent and Multi-Agent Systems 19 (3), 297–331.
[32] Brooks, R. A., 1986. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation 2 (1), 14–23.
[33] Burfoot, D., Pineau, J., Dudek, G., 2006. RRT-plan: a randomized algorithm for strips planning. In: AAAI. pp. 362–365.
[34] Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., Thrun, S., 1998. The interactive museum tour-guide robot. In: AAAI. pp. 11–18.
[35] Busoniu, L., Munos, R., De Schutter, B., Babuska, R., 2011. Optimistic planning for sparsely stochastic systems. In: ADPRL. pp. 48–55.
[36] Cambon, S., Alami, R., Gravot, F., 2009. A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. IJRR 28 (1), 104–126.
[37] Carbone, A., Finzi, A., Orlandini, A., Pirri, F., Nov. 2007. Model-based control architecture for attentive robots in rescue scenarios. Autonomous Robots 24 (1), 87–120.
[38] Ceballos, A., Bensalem, S., Cesta, A., de Silva, L., Fratini, S., Ingrand, F., Ocón, J., Orlandini, A., Py, F., Rajan, K., 2011. A Goal-Oriented Autonomous Controller for Space Exploration. In: ASTRA.
[39] Cesta, A., Finzi, A., Fratini, S., Orlandini, A., Tronci, E., 2010. Validation and verification issues in a timeline-based planning system. Knowl Eng Rev 25 (3), 299–318.
[40] Chatilla, R., Alami, R., Degallaix, B., Laruelle, H., 1992. Integrated planning and execution control of autonomous robot actions. In: ICRA.
[41] Chaumette, F., Hutchinson, S., 2008. Visual Servoing and Visual Tracking. In: Springer Handbook of Robotics. Springer, pp. 563–583.
[42] Chernova, S., Thomaz, A. L., 2014. Robot Learning from Human Teachers. Synthesis Lectures on AI and ML. Morgan and Claypool.
[43] Chien, S., Knight, R., Stechert, A., Sherwood, R., Rabideau, G., 2000. Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. In: AAAI.
[44] Choi, J., Amir, E., 2009. Combining Planning and Motion Planning. In: ICRA.
[45] Choset, H., Nagatani, K., 2001. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. IEEE Transactions on Robotics and Automation 17 (2), 125–137.
[46] Claßen, J., Röger, G., Lakemeyer, G., Nebel, B., 2012. Platas—Integrating Planning and the Action Language Golog. KI-Künstliche Intelligenz 26 (1), 61–67.
[47] Coates, A., Abbeel, P., Ng, A. Y., 2009. Apprenticeship learning for helicopter control. Communication ACM 52 (7), 97–105.
[48] Coles, A., Fox, M., Long, D., Smith, A., 2008. Planning with Problems Requiring Temporal Coordination. In: AAAI. pp. 892–897.
[49] Coles, A., Smith, A., 2007. Marvin: A heuristic search planner with online macro-action learning. JAIR 28, 119–156.
[50] Coles, A. J., Coles, A., Fox, M., Long, D., 2012. COLIN: Planning with Continuous Linear Numeric Change. JAIR 44, 1–96.
[51] Conrad, P., Shah, J., Williams, B. C., 2009. Flexible execution of plans with choice. In: ICAPS.
[52] Conrad, P., Williams, B. C., 2011. Drake: An Efficient Executive for Temporal Plans with Choice. JAIR.
[53] Coradeschi, S., Saffiotti, A., 2002. Perceptual anchoring: a key concept for plan execution in embedded systems. In: Beetz, M., Hertzberg, J., Ghallab, M., Pollack, M. E. (Eds.), Advances in Plan-Based Control of Robotic Agents. Springer-Verlag, pp. 89–105.
[54] Coradeschi, S., Saffiotti, A., 2003. An introduction to the anchoring problem. Robotics and Autonomous Systems 43 (2-3), 85–96.
[55] Coste-Maniere, E., Espiau, B., Rutten, E., 1992. A task-level robot programming language and its reactive execution. In: ICRA.
[56] Dean, T., Kanazawa, K., 1989. A model for reasoning about persistence and causation. Computational Intell 5 (3), 142–150.
[57] Dearden, R., Friedman, N., Andre, D., 1999. Model based Bayesian exploration. In: UAI. pp. 150–159.
[58] Dechter, R., Meiri, I., Pearl, J., 1991. Temporal constraint networks. Artificial Intelligence 49 (1-3), 61–95.
[59] Despouys, O., Ingrand, F., 1999. Propice-Plan: Toward a Unified Framework for Planning and Execution. In: ECP.
[60] Dietterich, T., 2000. Hierarchical reinforcement learning with MAXQ value function. JAIR 13, 227–303.
[61] Doherty, P., Granlund, G., Kuchcinski, K., Sandewall, E., Nordberg, K., Skarman, E., Wiklund, J., 2000. The WITAS unmanned aerial vehicle project. In: ECAI. pp. 747–755.
[62] Doherty, P., Haslum, P., Heintz, F., Merz, T., Nyblom, P., Persson, T., Wingman, B., 2004. A Distributed Architecture for Autonomous Unmanned Aerial Vehicle Experimentation. In: DARS.
[63] Doherty, P., Kvarnström, J., Heintz, F., Feb. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. Autonomous Agent and Multi-Agent Systems 19 (3), 332–377.
[64] Domshlak, C., Karpas, E., Markovitch, S., 2012. Online Speedup Learning for Optimal Planning. JAIR 44, 709–755.
[65] Dousson, C., Gaborit, P., Ghallab, M., 1993. Situation recognition: Representation and algorithms. IJCAI 13, 166–166.
[66] Dousson, C., Le Maigat, P., 2007. Chronicle Recognition Improvement Using Temporal Focusing and Hierarchization. In: IJCAI.
[67] Džeroski, S., De Raedt, L., Driessens, K., 2001. Relational reinforcement learning. Machine Learning 43 (1), 7–52.
[68] Effinger, R. T., Williams, B. C., Hofmann, A., Jun. 2010. Dynamic Execution of Temporally and Spatially Flexible Reactive Programs. In: AAAI Workshop: Bridging the Gap between Task and Motion Planning. pp. 1–8.
[69] Estlin, T., Gaines, D., Chouinard, C., Castano, R., Bornstein, B., Judd, M., Nesnas, I. A., Anderson, R., 2007. Increased Mars Rover Autonomy using AI Planning, Scheduling and Execution. In: ICRA. pp. 4911–4918.
[70] Fernández, F., García, J., veloso, M. M., 2010. Probabilistic Policy Reuse for inter-task transfer learning. Robotics and Autonomous Systems 58 (7), 866–871.
[71] Ferrein, A., Lakemeyer, G., 2008. Logic-based robot control in highly dynamic domains. Robotics and Autonomous Systems 56 (11), 980–991.
[72] Fichtner, M., Großmann, A., Thielscher, M., 2003. Intelligent execution monitoring in dynamic environments. Fundamenta Informaticae 57 (2-4), 371–392.
[73] Fikes, R. E., Aug. 1971. Monitored Execution of Robot Plans Produced by STRIPS. In: IFIP Congress. Ljubljana, Yugoslavia.
[74] Finzi, A., Ingrand, F., Muscettola, N., 2004. Model-based Executive Control through Reactive Planning for Autonomous Rovers. In: IROS.
[75] Firby, R. J., 1987. An investigation into reactive planning in complex domains. In: AAAI.
[76] Fleury, S., Herrb, M., Chatilla, R., 1997. GenoM: A Tool for the Specification and the Implementation of Operating Modules in a Distributed

Robot Architecture. In: IROS. Grenoble, France, pp. 842–848.
[77] Fox, M., Gerevini, A., Long, D., Serina, I., 2006. Plan stability: Replanning versus plan repair. In: ICAPS.
[78] Fox, M., Ghallab, M., Infantes, G., Long, D., 2006. Robot introspection through learned hidden markov models. Artificial Intelligence 170 (2), 59–113.
[79] Frank, J., Jónsson, A. K., 2003. Constraint-Based Attribute and Interval Planning. Constraints 8 (4).
[80] Fraser, G., Steinbauer, G., Wotawa, F., 2005. Plan execution in dynamic environments. In: Lecture Notes in Computer Science. Innovations in Applied Artificial Intelligence, pp. 208–217.
[81] Fratini, S., Cesta, A., De Benedictis, R., Orlandini, A., Rasconi, R., 2011. APSI-based deliberation in Goal Oriented Autonomous Controllers. In: ASTRA.
[82] Fusier, F., Valentin, V., Bremond, F., Thonnat, M., Borg, M., Thirde, D., Ferryman, J., 2007. Video understanding for complex activity recognition. Machine Vision and Applications 18 (3-4), 167–188.
[83] Galindo, C., Fernandez-Madrigal, J.-A., Gonzalez, J., Saffiotti, A., 2008. Robot Task Planning Using Semantic Maps. Robotics and Autonomous Systems 56 (11), 955–966.
[84] Galindo, C., Fernandez-Madrigal, J.-A., Gonzalez, J., Saffiotti, A., Buschka, P., 2007. Life-long optimization of the symbolic model of indoor environments for a mobile robot. IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics 37 (5), 1290–1304.
[85] Gat, E., 1997. On Three-Layer Architectures. In: Kortenkamp, D., Bonnasso, R. P., Murphy, R. (Eds.), Artificial Intelligence and Mobile Robots. AAAI Press.
[86] Geffner, H., Bonet, B., 2013. A Concise Introduction to Models and Methods for Automated Planning. Morgan & Claypool.
[87] Geib, C., Goldman, R. P., 2009. A probabilistic plan recognition algorithm based on plan tree grammars. Artificial Intelligence 173, 1101–1132.
[88] Ghallab, M., 1996. On Chronicles: Representation, On-line Recognition and Learning. In: KR. pp. 597–606.
[89] Ghallab, M., Laruelle, H., 1994. Representation and Control in IxTeT, a Temporal Planner. In: AIPS. pp. 61–67.
[90] Ghallab, M., Mounir-Alaoui, A., 1989. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. In: IJCAI.
[91] Ghallab, M., Nau, D. S., Traverso, P., Oct. 2004. Automated Planning: Theory and Practice. Morgann Kaufmann.
[92] Ghallab, M., Nau, D. S., Traverso, P., 2014. The actor's view of automated planning and acting: A position paper. Artificial Intelligence 208, 1–17.
[93] Gosavi, A., 2009. Reinforcement Learning: A Tutorial Survey and Recent Advances. INFORMS on Computing 21 (2), 178–192.
[94] Hähnel, D., Burgard, W., Lakemeyer, G., 1998. GOLEX—bridging the gap between logic (GOLOG) and a real robot. In: KI Advances in Artificial Intelligence. Springer, pp. 165–176.
[95] Hansen, E. A., Zilberstein, S., 2001. LAO*: A heuristic search algorithm that finds solutions with loops. Artificial Intelligence 129 (1), 35–62.
[96] Hartanto, R., Hertzberg, J., 2008. Fusing DL Reasoning with HTN Planning. In: KI: Advances in Artificial Intelligence. LNCS, Springer, pp. 62–69.
[97] Hauser, K., Latombe, J.-C., 2009. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In: ICAPS Workshop on Bridging the gap between task and motion planning.
[98] Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., Boutilier, C., 1998. Hierarchical solution of Markov decision processes using macro-actions. In: UAI. pp. 220–229.
[99] Hawes, N., 2011. A survey of motivation frameworks for intelligent systems. Artificial Intelligence 175 (5), 1020–1036.
[100] Hayes-Roth, B., Jul. 1985. A Blackboard Architecture for Control. Artificial Intelligence 26 (3), 251–321.
[101] Heintz, F., Kvarnström, J., Doherty, P., 2010. Bridging the sense-reasoning gap: DyKnow-Stream-based middleware for knowledge processing. Advanced Engineering Informatics 24 (1), 14–26.
[102] Helmert, M., 2009. Concise finite-domain representations for PDDL planning tasks. Artificial Intelligence 173 (5-6), 503–535.
[103] Hongeng, S., Nevatia, R., Bremond, F., 2004. Video-based event recognition: activity representation and probabilistic recognition methods. Computer Vision and Image Understanding 96 (2), 129–162.
[104] Infantes, G., Ghallab, M., Ingrand, F., Oct. 2010. Learning the behavior model of a robot. Autonomous Robots, 1–21.
[105] Ingham, M. D., Ragno, R. J., Williams, B. C., 2001. A Reactive Model-based Programming Language for Robotic Space Explorers. In: ISAIRAS.
[106] Ingrand, F., Chatilla, R., Alami, R., Robert, F., 1996. PRS: a high level supervision and control language for autonomous mobile robots. In: ICRA. pp. 43–49.
[107] Ingrand, F., Ghallab, M., Jan. 2014. Robotics and Artificial Intelligence: a Perspective on Deliberation Functions. AI Communications 27 (1), 63–80.
[108] Ingrand, F., Lacroix, S., Lemai-Chenevier, S., Py, F., 2007. Decisional autonomy of planetary rovers. JFR 24 (7), 559–580.
[109] Jiménez, S., de La Rosa, T., Fernández, S., Fernández, F., Borrajo, D., 2012. A Review of Machine Learning for Automated Planning. The Knowledge Engineering Review 27 (4), 433–467.
[110] Jónsson, A. K., Morris, P. H., Muscettola, N., Rajan, K., Smith, B. D., 2000. Planning in Interplanetary Space: Theory and Practice. In: AIPS. pp. 177–186.
[111] Judah, K., Fern, A. P., Dietterich, T. G., 2012. Active Imitation Learning via Reduction to IID Active Learning. In: UAI. pp. 428–437.
[112] Kaelbling, L. P., Littman, M. L., Moore, A. W., 1996. Reinforcement Learning: A Survey. JAIR 4.
[113] Kaelbling, L. P., Lozano-Perez, T., 2011. Hierarchical task and motion planning in the now. In: ICRA. pp. 1470–1477.
[114] Kanoun, O., Laumond, J.-P., Yoshida, E., 2011. Planning foot placements for a humanoid robot: A problem of inverse kinematics. IJRR 30 (4), 476–485.
[115] Karlsson, L., Bouguerra, A., Broxvall, M., Coradeschi, S., Saffiotti, A., 2008. To Secure an Anchor – A recovery planning approach to ambiguity in perceptual anchoring. AI Communications 21 (1), 1–14.
[116] Kautz, H. A., Allen, J. F., 1986. Generalized plan recognition. In: AAAI. pp. 32–37.
[117] Kearns, M., Mansour, Y., Ng, A. Y., 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes.

Machine Learning 49, 193–208.
[118] Khoo, A., Horswill, I., 2003. Grounding inference in distributed multi-robot environments. Robotics and Autonomous Systems 43, 121–132.
[119] Knight, R., Rabideau, G., Chien, S., Engelhardt, B., Sherwood, R., 2001. Casper: space exploration through continuous planning. Intelligent Systems, IEEE 16 (5), 70–75.
[120] Kober, J., Bagnell, J. A., Peters, J., Aug. 2013. Reinforcement Learning in Robotics: A Survey. IJRR 32 (11), 1238–1274.
[121] Kober, J., Peters, J., 2011. Policy search for motor primitives in robotics. Machine Learning 84 (1), 171–203.
[122] Kolobov, A., Mausam, Weld, D., 2010. SixthSense: Fast and Reliable Recognition of Dead Ends in MDPs. In: AAAI.
[123] Kolobov, A., Mausam, Weld, D., 2012. LRTDP vs. UCT for Online Probabilistic Planning. In: AAAI.
[124] Kolobov, A., Mausam, Weld, D., Geffner, H., 2011. Heuristic search for generalized stochastic shortest path MDPs. In: ICAPS.
[125] Konolige, K., Marder-Eppstein, E., Marthi, B., 2011. Navigation in Hybrid Metric-Topological Maps. In: ICRA.
[126] Konolige, K., Myers, K. L., Ruspini, E. H., Saffiotti, A., Apr. 1997. The Saphira architecture: a design for autonomy. JETAI 9 (2-3), 215–235.
[127] Kortenkamp, D., Simmons, R., 2008. Robotic Systems Architectures and Programming. In: Siciliano, B., Khatib, O. (Eds.), Springer Handbook of Robotics. Springer, pp. 187–206.
[128] Kroemer, O., Peters, J., 2011. Active exploration for robot parameter selection in episodic reinforcement learning. In: ADPRL. pp. 25–31.
[129] Krüger, V., Kragic, D., Ude, A., Geib, C., 2007. The meaning of action: a review on action recognition and mapping. Advanced Robotics 21 (13), 1473–1501.
[130] Kruse, T., Pandey, A. K., Alami, R., Kirsch, A., 2013. Human-aware robot navigation: A survey. Robotics and Autonomous Systems 61 (12), 1726–1743.
[131] Kuipers, B., Byun, Y.-T., 1991. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. Robotics and Autonomous Systems 8 (1-2), 47–63.
[132] Kuipers, B., Modayil, J., Beeson, P., MacMahon, M., Savelli, F., 2004. Local Metrical and Global Topological Maps in the Hybrid Spatial Semantic Hierarchy. In: ICRA. pp. 4845–4851.
[133] Kvarnström, J., Doherty, P., 2000. TALplanner: A temporal logic based forward chaining planner. Annals of Mathematics and Artificial Intelligence 30 (1), 119–169.
[134] Lamine, K., Kabanza, F., 2000. History checking of temporal fuzzy logic formulas for monitoring behavior-based mobile robots. In: ICTAI.
[135] Laporte, C., Arbel, T., 2006. Efficient discriminant viewpoint selection for active bayesian recognition. IJRR 68 (3), 267–287.
[136] LaValle, S. M. (Ed.), 2006. Planning Algorithms. Cambridge University Press.
[137] Lemai-Chenevier, S., Ingrand, F., 2004. Interleaving Temporal Planning and Execution in Robotics Domains. In: AAAI.
[138] Lemaignan, S., Espinoza, R. R., Mösenlechner, L., Alami, R., Beetz, M., 2010. ORO, a knowledge management platform for cognitive architectures in robotics. In: IROS.
[139] Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., Scherl, R. B., 1997. GOLOG: A logic programming language for dynamic domains. The Journal of Logic Programming 31 (1), 59–83.
[140] Levine, S. J., Williams, B. C., Nov. 2014. Concurrent Plan Recognition and Execution for Human-Robot Teams. In: ICAPS.
[141] Likhachev, M., Gordon, G., Thrun, S., 2004. Planning for markov decision processes with sparse stochasticity. Advances in Neural Information Processing Systems (NIPS) 17.
[142] Magazzeni, D., Py, F., Fox, M., Long, D., Rajan, K., 2013. Policy learning for autonomous feature tracking. Autonomous Robots 37 (1), 47–69.
[143] Magnenat, S., Chappelier, J. C., Mondada, F., 2012. Integration of Online Learning into HTN Planning for Robotic Tasks. In: AAAI Spring Symposium.
[144] Marthi, B. M., Kaelbling, L. P., Lozano-Perez, T., 2007. Learning hierarchical structure in policies. In: NIPS Hierarchical Organization of Behavior Workshop.
[145] Martinez-Cantin, R., Freitas, N., Brochu, E., Castellanos, J., Doucet, A., Aug. 2009. A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. Autonomous Robots 27 (2), 93–103.
[146] Mausam, Kolobov, A., 2012. Planning with Markov Decision Processes: An AI Perspective. Morgan & Claypool.
[147] McGann, C., Py, F., Rajan, K., Henthorn, R., McEwen, R., 2008. A deliberative architecture for AUV control. In: ICRA. pp. 1049–1054.
[148] Mericli, C., veloso, M. M., Akin, H. L., 2012. Multi-resolution Corrective Demonstration for Efficient Task Execution and Refinement. International Journal of Social Robotics 4 (4), 423–435.
[149] Moeslund, T. B., Hilton, A., Krüger, V., 2006. A survey of advances in vision-based human motion capture and analysis. Computer Vision and Image Understanding 104 (2-3), 90–126.
[150] Molineaux, M., Klenk, M., Aha, D., 2010. Goal-driven autonomy in a Navy strategy simulation. In: AAAI. pp. 1548–1554.
[151] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., 2003. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In: IJCAI. pp. 1151–1156.
[152] Morisset, B., Ghallab, M., 2008. Learning how to combine sensory-motor functions into a robust behavior. Artificial Intelligence 172 (4-5), 392–412.
[153] Morris, P. H., Muscettola, N., 2005. Temporal Dynamic Controllability Revisited. In: AAAI. pp. 1193–1198.
[154] Morris, P. H., Muscettola, N., Vidal, T., 2001. Dynamic Control of Plans with Temporal Uncertainty. In: IJCAI. pp. 494–502.
[155] Muscettola, N., Dorais, G., Fry, C., Levinson, R., Plaunt, C., 2000. A Unified Approach to Model-Based Planning and Execution. In: IAS.
[156] Muscettola, N., Dorais, G., Fry, C., Levinson, R., Plaunt, C., 2002. IDEA: Planning at the Core of Autonomous Reactive Agents. In: IWPSS.
[157] Muscettola, N., Nayak, P. P., Pell, B., Williams, B. C., 1998. Remote Agent: to boldly go where no AI system has gone before. Artificial Intelligence 103, 5–47.
[158] Myers, K. L., 1996. A procedural knowledge approach to task-level control. In: AAAI.
[159] Myers, K. L., 1999. CPEF: Continuous Planning and Execution Framework. AI Magazine 20 (4), 63–69.
[160] Nebel, B., Koehler, J., 1995. Plan Reuse Versus Plan Generation: A Theoretical and Empirical Analysis. Artificial Intelligence 76 (1-2), 427–454.

[161] Nesnas, I. A., Wright, A., Bajracharya, M., Simmons, R., Estlin, T., Oct. 2003. CLARAty and Challenges of Developing Interoperable Robotic Software. In: IROS.
[162] Newton, M. A. H., Levine, J., Fox, M., Long, D., 2007. Learning macro-actions for arbitrary planners and domains. In: ICAPS.
[163] Ng, A., Russel, S. J., 2000. Algorithms for inverse reinforcement learning. In: International Conference on Machine Learning. pp. 1–8.
[164] Nguyen-Tuong, D., Peters, J., 2011. Model learning for robot control: a survey. Cognitive Processing 12 (4), 319–340.
[165] Nilsson, M., Kvarnström, J., Doherty, P., 2014. Incremental Dynamic Controllability in Cubic Worst-Case Time. In: International Symposium on Temporal Representation and Reasoning.
[166] Ong, S. C. W., Png, S. W., Hsu, D., Lee, W. S., 2010. Planning under Uncertainty for Robotic Tasks with Mixed Observability. IJRR 29 (8), 1053–1068.
[167] Pecora, F., Cirillo, M., Dell'Osa, F., Ullberg, J., Saffiotti, A., 2012. A constraint-based approach for proactive, context-aware human support. Journal of Ambient Intelligence and Smart Environments 4 (4), 347–367.
[168] Petrick, R., Bacchus, F., 2004. Extending the Knowledge-Based approach to Planning with Incomplete Information and Sensing. In: ICAPS. pp. 2–11.
[169] Pettersson, O., 2005. Execution monitoring in robotics: A survey. Robotics and Autonomous Systems 53 (2), 73–88.
[170] Pettersson, O., Karlsson, L., Saffiotti, A., 2003. Model-free execution monitoring in behavior-based mobile robotics. In: ICAR.
[171] Pineau, J., Montemerlo, M., Pollack, M. E., Roy, N., Thrun, S., Mar. 2003. Towards robotic assistants in nursing homes: Challenges and results. Robotics and Autonomous Systems 42 (3-4), 271–281.
[172] Plaku, E., Hager, G., 2009. Sampling-based Motion Planning with Symbolic, Geometric, and Differential Constraints. In: ICAPS Workshop on Bridging the gap between task and motion planning.
[173] Pollack, M. E., Horty, J., 1999. There's more to life than making plans: plan management in dynamic, multiagent environments. AI Magazine 20 (4), 71.
[174] Powell, J., Molineaux, M., Aha, D., 2011. Active and Interactive Discovery of Goal Selection Knowledge. In: FLAIRS.
[175] Prentice, S., Roy, N., 2009. The belief roadmap: Efficient planning in belief space by factoring the covariance. IJRR 28 (11–12), 1448–1465.
[176] Py, F., Rajan, K., McGann, C., 2010. A Systematic Agent Framework for Situated Autonomous Systems. In: AAMAS. pp. 583–590.
[177] Pynadath, D. V., Wellman, M. P., 2000. Probabilistic State-Dependent Grammars for Plan Recognition. In: UAI. pp. 507–514.
[178] Rabideau, G., Knight, R., Chien, S., Fukunaga, A., Govindjee, A., 1999. Iterative Repair Planning for Spacecraft Operations in the ASPEN System. In: ISAIRAS.
[179] Rabiner, L., Juang, B. H., 1986. An introduction to hidden Markov models. ASSP Magazine, IEEE 3 (1), 4–16.
[180] Rajan, K., Py, F., 2012. T-REX: Partitioned inference for AUV mission control. In: Roberts, G. N., Sutton, R. (Eds.), Further Advances in Unmanned Marine Vehicles. The Institution of Engineering and Technology, pp. 171–199.
[181] Rajan, K., Py, F., Barreiro, J., Nov. 2012. Towards Deliberative Control in Marine Robotics. In: Marine Robot Autonomy. Springer New York, New York, NY, pp. 91–175.
[182] Ramirez, M., Geffner, H., 2010. Probabilistic plan recognition using off-the-shelf classical planners. In: AAAI. pp. 1121–1126.
[183] Ranganathan, A., Dellaert, F., 2007. Semantic modeling of places using objects. In: RSS. pp. 27–30.
[184] Ranganathan, A., Dellaert, F., 2011. Online probabilistic topological mapping. The International Journal of Robotics Research 30 (6), 755–771.
[185] Rintanen, J., 2000. An Iterative Algorithm for Synthesizing Invariants. In: AAAI. pp. 806–811.
[186] Rockel, S., Neumann, B., Zhang, J., Dubba, K. S. R., Cohn, A. G., Konecný, Š., Mansouri, M., Pecora, F., Saffiotti, A., Gunther, M., Stock, S., Hertzberg, J., Tomé, A. M., Pinho, A., Seabra Lopes, L., von Riegen, S., Hotz, L., 2013. An Ontology-Based Multi-Level Robot Architecture for Learning from Experiences. In: AAAI Spring Symposium. pp. 1–6.
[187] Rosenthal, S., veloso, M. M., Dey, A. K., 2011. Task Behavior and Interaction Planning for a Mobile Service Robot that Occasionally Requires Help. In: AAAI WS: Automated Action Planning for Autonomous Mobile Robots.
[188] Ruml, W., Do, M. B., Zhou, R., Fromherz, M. P., 2011. On-line planning and scheduling: An application to controlling modular printers. JAIR 40 (1), 415–468.
[189] Russel, S. J., Norvig, P., Mar. 2007. Statistical Learning Methods. In: Artificial Intelligence: A Modern Approach. Prentice Hall.
[190] Samadi, M., Kollar, T., veloso, M. M., 2012. Using the web to interactively learn to find objects. In: AAAI. pp. 2074–2080.
[191] Sandewall, E., 1995. Features and Fluents. Oxford university Press.
[192] Santana, P. H. R. Q. A., Williams, B. C., Nov. 2014. Chance-Constrained Consistency for Probabilistic Temporal Plan Networks. In: ICAPS.
[193] Schermerhorn, P. W., Kramer, J. F., Middendorff, C., Scheutz, M., 2006. DIARC: A Testbed for Natural Human-Robot Interaction. In: AAAI. pp. 1972–1973.
[194] Schoppers, M. J., 1987. Universal plans for reactive robots in unpredictable environments. In: IJCAI.
[195] Selen, L. P. J., Shadlen, M. N., Wolpert, D. M., 2012. Deliberation in the Motor System: Reflex Gains Track Evolving Evidence Leading to a Decision. Journal of Neuroscience 32 (7), 2276–2286.
[196] Shapiro, S. C., Ismail, H. O., 2003. Anchoring in a grounded layered architecture with integrated reasoning. Robotics and Autonomous Systems 43, 97–108.
[197] Sigaud, O., Peters, J. (Eds.), 2010. From Motor Learning to Interaction Learning in Robots. Vol. 264 of Studies in Computational Intelligence. Springer.
[198] Siméon, T., Laumond, J.-P., Cortés, J., Sahbani, A., 2004. Manipulation Planning with Probabilistic Roadmaps. IJRR 23 (7-8), 729–746.
[199] Simmons, R., 1992. Concurrent planning and execution for autonomous robots. Control Systems, IEEE 12 (1), 46–50.
[200] Simmons, R., 1994. Structured control for autonomous robots. IEEE Transactions on Robotics and Automation 10 (1), 34–43.
[201] Simmons, R., Apfelbaum, D., 1998. A task description language for robot control. In: IROS. pp. 1931–1937 vol.3.
[202] Smith, D. E., Weld, D., 1999. Temporal Planning with Mutual Exclusion Reasoning. In: IJCAI.
[203] Sridharan, M., Wyatt, J. L., Dearden, R., 2008. HiPPo: Hierarchical POMDPs for Planning Information Processing and Sensing Actions on a Robot. In: ICAPS. pp. 346–354.
[204] Stulp, F., Beetz, M., 2008. Refining the execution of abstract actions with learned action models. JAIR 32 (1), 487–523.

[205] Sutton, R. S., Barto, A. G., 1998. Reinforcement Learning: An Introduction. MIT Press.
[206] Teichteil-Königsbuch, F., Kuter, U., Infantes, G., 2010. Incremental plan aggregation for generating policies in MDPs. In: AAMAS.
[207] Teichteil-Königsbuch, F., Lesire, C., Infantes, G., 2011. A generic framework for anytime execution-driven planning in robotics. In: ICRA.
[208] Tenorth, M., Beetz, M., 2013. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. The International Journal of Robotics Research 32 (5), 566–590.
[209] Thrun, S., 2002. Robotic Mapping: A Survey. In: Lakemeyer, G., Nebel, B. (Eds.), Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann.
[210] Tonino, H., Bos, A., de Weerdt, M., Witteveen, C., 2002. Plan coordination by revision in collective agent based systems. Artificial Intelligence 142 (2), 121–145.
[211] Trevizan, F. W., veloso, M. M., 2012. Short-sighted stochastic shortest path problems. In: ICAPS.
[212] van der Krogt, R., de Weerdt, M., 2005. Plan repair as an extension of planning. In: ICAPS.
[213] Vattam, S., Klenk, M., Molineaux, M., Aha, D. W., 2013. Breadth of Approaches to Goal Reasoning: A Research Survey. In: Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning.
[214] Velez, J., Hemann, G., Huang, A., Posner, I., Roy, N., 2011. Planning to perceive: Exploiting mobility for robust object detection. In: ICAPS.
[215] veloso, M. M., Rizzo, P., 1998. Mapping planning actions and partially-ordered plans into execution knowledge. In: Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments. pp. 94–97.
[216] Verma, V., Estlin, T., Jónsson, A. K., Pasareanu, C., Simmons, R., Tso, K., 2005. Plan execution interchange language (PLEXIL) for executable plans and command sequences. In: IWPSS.
[217] Vidal, T., Ghallab, M., 1996. Dealing with uncertain durations in temporal constraints networks dedicated to planning. In: ECAI. pp. 48–52.
[218] Waibel, M., Beetz, M., Civera, J., D'Andrea, R., Elfring, J., Galvez-Lopez, D., Haussermann, K., Janssen, R., Montiel, J. M. M., Perzylo, A., Schiessle, B., Tenorth, M., Zweigle, O., van de Molengraft, R., 2011. RoboEarth. IEEE RAM 18 (2), 69–82.
[219] Walsh, T. J., Goschin, S., Littman, M. L., 2010. Integrating sample-based planning and model-based reinforcement learning. In: AAAI.
[220] Walsh, T. J., Littman, M. L., 2008. Efficient learning of action schemas and web-service descriptions. In: AAAI.
[221] Wang, F. Y., Kyriakopoulos, K. J., Tsolkas, A., Saridis, G. N., 1991. A Petri-net coordination model for an intelligent mobile robot. IEEE Transactions on Systems, Man, and Cybernetics 21 (4), 777–789.
[222] Wilkins, D. E., 1988. Practical Planning. Extending the Classical AI Planning Paradigm. Morgan Kaufman.
[223] Wilkins, D. E., Myers, K. L., 1995. A common knowledge representation for plan generation and reactive execution. Journal of Logic and Computation 5 (6), 731–761.
[224] Williams, B. C., Abramson, M., 2001. Executing Reactive, Model-based Programs through Graph-based Temporal Planning. In: IJCAI.
[225] Williams, B. C., Gupta, V., 1999. Unifying model-based and reactive programming within a model-based executive. In: Workshop on Principles of Diagnosis.
[226] Williams, B. C., Ingham, M. D., 2003. Model-based Programming of Intelligent Embedded Systems and Robotic Space Explorers. Proc. of the IEEE: Special Issue on Modeling and Design of Embedded Software 91 (1), 212–237.
[227] Williams, B. C., Nayak, P. P., 1996. A Model-based Approach to Reactive Self-Configuring Systems. In: AAAI. pp. 971–978.
[228] Williams, B. C., Nayak, P. P., 1997. A reactive planner for a model-based executive. In: IJCAI.
[229] Wilson, A., Fern, A., Tadepalli, P., 2012. A Bayesian Approach for Policy Learning from Trajectory Preference Queries. Advances in neural information processing systems, 1142–1150.
[230] Wilson, M. A., McMahon, J., Aha, D. W., May 2014. Bounded Expectations for Discrepancy Detection in Goal-Driven Autonomy. In: AAAI Workshop on AI and Robotics.
[231] Wolfe, J., Marthi, B., Russel, S. J., 2010. Combined task and motion planning for mobile manipulation. In: ICAPS. pp. 254–258.
[232] Wu, Y., Huang, T. S., 1999. Vision-based gesture recognition: A review. In: Braffort, A., Gherbi, R., Gibet, S., Teil, D., Richardson, J. (Eds.), Gesture-Based Communication in Human-Computer Interaction. Springer, pp. 103–115.
[233] Xu, Y., Fern, A., , Yoon, S. W., 2007. Discriminative learning of beam-search heuristics for planning. In: IJCAI.
[234] Yang, Q., 1997. Intelligent Planning: A Decomposition and Abstraction Based Approach. Springer-Verlag.
[235] Yang, Q., Wu, K., Jiang, Y., 2007. Learning action models from plan examples using weighted MAX-SAT. Artificial Intelligence 171 (2), 107–143.
[236] Yoon, S., Fern, A., Givan, R., 2006. Learning heuristic functions from relaxed plans. In: ICAPS.
[237] Younes, H., Simmons, R., 2003. VHPOP: Versatile heuristic partial order planner. JAIR 20 (1), 405–430.
[238] Zickler, S., veloso, M. M., 2009. Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In: AAMAS.
[239] Zilberstein, S., Washington, R., Bernstein, D. S., Mouaddib, A.-I., Feb. 2003. Decision-Theoretic Control of Planetary Rovers. In: Plan-Based Control of Robotic Agent. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 270–289.
[240] Zimmerman, T., Kambhampati, S., 2003. Learning-Assisted Automated Planning: Looking Back, Taking Stock, Going Forward. AI Magazine 24 (2), 73.
[241] Ziparo, V. A., Iocchi, L., Lima, P. U., Nardi, D., Palamara, P. F., 2011. Petri net plans. Autonomous Agent and Multi-Agent Systems 23 (3), 344–383.