Robotics and Artificial Intelligence: a Perspective on Deliberation Functions

Félix Ingrand, Malik Ghallab

Robotics and Artificial Intelligence: a Perspective on Deliberation Functions

Félix Ingrand Malik Ghallab
LAAS–CNRS, Université de Toulouse
7, Av. Colonel Roche, 31077 Toulouse, France
E-mail: {felix,malik}@laas.fr

April 3, 2015

Abstract

Abstract: Despite a very strong synergy between Robotics and AI at their early beginning, the two fields progressed widely apart in the following decades. However, we are witnessing a revival of interest in the fertile domain of embodied machine intelligence, which is due in particular to the dissemination of more mature techniques from both areas and more accessible robot platforms with advanced sensory motor capabilities, and to a better understanding of the scientific challenges of the AI–Robotics intersection.
The ambition of this paper is to contribute to this revival. It proposes an overview of problems and approaches to autonomous deliberate action in robotics. The paper advocates for a broad understanding of deliberation functions. It presents a synthetic perspective on *planning, acting, perceiving, monitoring, goal reasoning* and their *integrative architectures*, which is illustrated through several contributions that addressed deliberation from the AI–Robotics point of view.

1 Introduction

Robotics is an interdisciplinary integrative field, at the confluence of several areas, ranging from mechanical and electrical engineering to control theory and computer science, with recent extensions toward material physics, bioengineering or cognitive sciences. The AI–Robotics intersection is very rich. It covers issues such as:

- deliberate action, planning, acting, monitoring and goal reasoning,
- perceiving, modeling and understanding open environments,
- interacting with human and other robots,
- learning models required by the above functions,
- integrating these functions in an adaptable and resilient architecture.

Robotics has always been a fertile inspiration paradigm for AI research, frequently referred to in its literature, in particular in the above topics. The early days of AI are rich in pioneering projects fostering a strong AI research agenda on robotics platforms. Typical examples are Shakey at SRI [85] and the Stanford Cart in the sixties, or Hilare at LAAS [36] and the CMU Rover [70] in the seventies. However, in the following decades the two fields developed in diverging directions; robotics expanded mostly outside of AI laboratories. Hopefully, a revival of the synergy between the two fields is currently being witnessed. This revival is due in particular to more mature techniques in robotics and AI, to the development of inexpensive robot platforms with more advanced sensing and control capabilities, to a number of popular competitions, and to a better understanding of the scientific challenges of machine intelligence, to which we would like to contribute here.

This revival is particularly strong in Europe where a large number of groups is actively contributing to the AI–Robotics interactions. For example, out of the 260 members of the Euron network,[1] about a third investigate robotics decision and cognitive functions. A similar ratio holds for the robotics projects in FP6 and FP7 (around a hundred). Many other european groups not within Euron and projects outside of EU programs are equally relevant to the AI and Robotics synergy. This focused perspective on *deliberative capabilities* in robotics cannot pay a fair tribute to all european actors of this synergy. It illustrates however several contributions from a few groups throughout Europe.[2] Its ambition is not to cover a comprehensive survey of deliberation issues, and even less of the AI–Robotics intersection. In the limited scope of this special issue, we propose a synthetic view of deliberation functions. We discuss the main problems involved in their development and exemplify a few approaches that addressed these problems. This "tour d'horizon" allows us to advocate for a broad and integrative view of deliberation, where problems are beyond search in planning, and beyond the open-loop triggering of commands in acting. We hope through this perspective to strengthen the AI–Robotics synergies.

The outline of the paper is the following: five deliberation functions are introduced in the next section; these are successively addressed through illustrative contributions; section 8 is devoted to architecture problems, followed by a conclusion.

2 Deliberation functions in robotics

Deliberation refers to purposeful, chosen or planned actions, carried out in order to achieve some objectives. Many robotics applications do not require deliberation capabilities, e.g., fixed robots in manufacturing and other well-modeled environments; vacuum cleaning and other devices limited to a single task; surgical and other tele-operated robots. Deliberation is a critical functionality for

[1] http://www.euron.org/

[2] e.g., from Barcelona, Bremen, Freiburg, Grenoble, Karlsruhe, London, Lille, Linköping, Munich, Örebro, Osnabrück, Oxford, Rennes, Roma and Toulouse

an *autonomous* robot facing a *variety of environments* and a *diversity of tasks*.

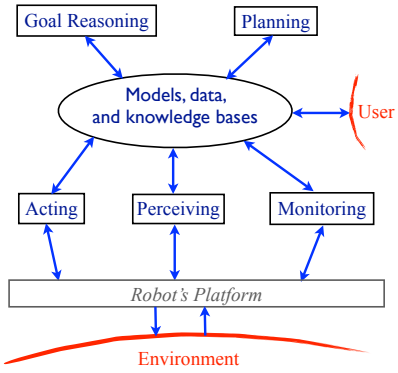

Figure 1: Schematic view of *deliberation functions*.

Several functions can be required for acting deliberately. The frontiers between these functions may depend on specific implementations and architectures, but it is clarifying to distinguish the following five *deliberation functions*, schematically depicted in figure 1:

- *Planning*: combines prediction and search to synthesize a trajectory in an abstract action space, using predictive models of feasible actions and of the environment.
- *Acting*: implements on-line close-loop feedback functions that process streams of sensors stimulus to actuators commands in order to refine and control the achievement of planned actions.
- *Perceiving*: extracts environment features to identify states, events, and situations relevant for the task. It combines bottom-up sensing, from sensors to meaningful data, with top-down focus mechanisms, sensing actions and planning for information gathering.
- *Monitoring*: compares and detects discrepancies between predictions and observations, performs diagnosis and triggers recovery actions.
- *Goal reasoning*: keeps current commitments and goals into perspective, assessing their relevance given observed evolutions, opportunities, constraints or failures, deciding about commitments to be abandoned, and goals to be updated.

These deliberation functions interact within a complex architecture (not depicted in Fig. 1) that will be discussed later. They are interfaced with the environment through the robot's *platform functions*, i.e., devices offering sensing and actuating capabilities, including signal processing and low-level control functions. The frontier between sensory-motor functions and deliberation functions depends on how variable are the environments and the tasks. For example, motion control along a predefined path is usually a platform function, but navigation to some destination requires one or several deliberation skills, integrating path planning, localization, collision avoidance, etc.

Learning capabilities change this frontier, e.g., in a familiar environment a navigation skill is compiled down into a low-level control with pre-cached parameters. A *metareasoning function* is also needed for trading off deliberation time for action time: critical tasks require careful deliberation, while less important or more urgent ones may not need, or allow for, more than fast approximate solutions, at least for a first reaction.[3]

3 Planning

Over the past decades, the field of automated planning achieved tremendous progress such as a speed up of few orders of magnitude in the performance of Strips-like classical planning, as well as numerous extensions in representations and improvements in algorithms for probabilistic and other non-classical planning [35]. Robotics stresses particular issues in automated planning, such as handling time and resources, or dealing with uncertainty, partial knowledge and open domains. Robots facing a variety of tasks need domain specific as well as domain independent task planners, whose correct integration remains a challenging problem.

Motion and manipulation planning are key capabilities for a robot, requiring specific representations for geometry, kinematics and dynamics. *Probabilistic Roadmaps* and *Rapid Random Trees* are well developed and mature techniques for motion planners that scale up efficiently and allow for numerous extensions [61]. The basic idea is to randomly sample the configuration space of the robot (i.e., the vector space of its kinematics parameters) into a graph where each vertex is a free configuration (away from obstacles) and each edge a direct link in the free space between two 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 [87]. Many other open problems remain in motion and manipulation planning, such as dynamics and stability constraints, e.g. for a humanoid robot [46], or visibility constraints to allow for visual servoing [14].

Task planning and motion/manipulation planning have been brought together in several work. The Asymov planner [12] combines a state-space planner with a search in the motion configuration space. It defines *places* which are both states, as well as sets of free configurations. Places define bridges between the two search spaces. The state-space search prunes a state whose corresponding set of free configurations does not meet current reachability conditions. Asymov has been extended to manipulation planning and to multi-robot planning of collaborative tasks, such as two robots assembling a table.

The integration of motion and task planning is also explored in [96] with Angelic Hierarchical Planning (AHP). AHP plans over sets of states with the

[3]Learning as well as metareasoning are certainly needed for deliberation; they are not covered here to keep the argument focused.

notion of *reachable set 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. A high-level action has several possible decompositions into primitives. A plan of high-level actions can be refined into the product of all feasible decompositions of its actions. A plan is acceptable if it has at least one feasible decomposition. Given such a plan, the robot chooses opportunistically a feasible decomposing for each high-level action (AHP refers to the angelic semantics of nondeterminism). The bounds used to characterize reachable sets of states are obtained by simulation of the primitives, including through motion and manipulation planning, for random values of the state variables.

A different coupling of a hierarchical task planner to *fast geometric suggesters* is developed in [45]. These suggesters are triggered when the search in the decomposition tree requires geometric information. They do not solve completely the geometric problem, but they provide information that allows the search to continue down to leaves of the tree. The system alternates between planning phases and execution of primitives, including motion and manipulation actions. Online planning allows to run motion or manipulation planners (not suggesters) in fully known states. The approach assumes that the geometric preconditions of the abstract actions can be computed quickly and efficiently by the suggesters, and that the sub-goals resulting from actions decomposition are executed in sequence (no parallelism). The resulting system is not complete. Failed actions should be reversible at a reasonable cost. For problems where these assumptions are met, the system is able to quickly produce correct plans.

4 Acting

In contrast to planning that can easily be specified as an offline predictive function, decoupled from the intricacies of the executing platform, acting is more difficult to define as a deliberation function. The frequent reference to *execution control* is often reductive: there is more to it than just triggering actions prescribed by a plan. Acting has to handle noisy sensors and imperfect models. It requires *non-deterministic*, *partially observable* and *dynamic* environment models, dealt with through *closed-loop commands*.

To integrate these requirements with those of predictive planning models, different forms of hierarchization are usually explored. For example (figure 2):

- planning deals with abstract preconditions-effects actions;
- acting refines opportunistically each action into *skills* and a skill further down into *commands*. This refinement mechanism may also use some planning techniques but with distinct state space and action space than those of the planner.

The skill into which an action is refined may change during that action execution. For example, several navigation skills may offer different localization or motion control capabilities adapted to different environment features. A `goto(room1)` action can be refined into a sequence of different navigation skills.

This hierarchization scheme may rely on distinct knowledge representations, e.g. STRIPS operators combined to PRS [42] or RAP [27] procedures. In

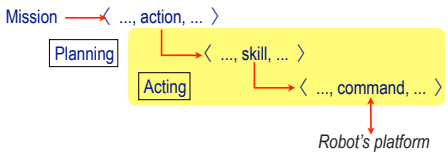

Figure 2: Refining actions into skills.

some cases a single representation is used for specifying planning and acting knowledge, e.g., Golog [63] or TAL [20] languages. Other approaches use a single representation seen at different levels of abstractions and refined appropriately, as in Hierarchical MDPs [38] for example.

Various computational techniques can be used to design a deliberate acting system. We propose to organize these approaches into five categories (see table 1) presented in the following subsections. Before discussing and illustrating these approaches, let us introduce the main functions needed to carry out a planned abstract action.

Refinement In most systems, the plan steps produced by the Planning component is not directly executable as a robot command. The `goto(room1)` action requires sending commands to the robot to perceive the environment, plan the path, execute it avoiding dynamic obstacles, etc. The refinement process needs be context dependent in order to select the most appropriate skills according to the online observation. It should be able to consider alternative refinements in case of failure.

Instantiation/Propagation Acting skills are often applicable to a range of situations. Their models use parameters whose value can be chosen at execution time or observed in the environment. Chosen or observed value have to be propagated in the rest of the skills down to the lowest level to issue commands.

Time management/Coordination Acting is performed in a close loop taking into account the dynamics of the environment. Adequate responses must be given in a timely manner. Some systems reason about time, deadlines as well as durations. Other systems handle a more symbolic representation of time with concurrency, rendez-vous and synchronization.

Handling nondeterminism and uncertainty Actions may have non-nominal effects. Furthermore, exogenous events in a dynamic environment are seldom predictable in a deterministic way. Finally, uncertainties in observations have to be taken into account.

Plan repair Some Acting approaches can repair the plan being executed. This is often performed using part of the problem search space already developed and explored by the Planner (hence, with an overlap between acting and planning). The idea is to solve new flaws or new threats that appeared during execution by minimizing the changes in the remaining of the plan. Even if repair may not be more efficient that replanning, there are cases where part of the plan has already been shared with other agents (robots or humans) which expect the robot to commit to it.

Another important aspect of Acting is how the skills are acquired and used. Are they completely hand written or learned? Are they used directly as programs or as specification models from which further synthesis is performed? Finally, there is the consistency verification issue between the Acting knowledge the Planning knowledge. We will see that some of the proposed formalism for representing skills are more adapted to validation and verification, even if this function is not always performed.

4.1 Procedure–based approaches

In procedure-based approaches, action refinement is done with handwritten skills. In RAP [27], each procedure is in charge of satisfying a particular goal, corresponding to a planned action. Deliberation chooses the appropriate procedure for the current state. The system commits to a goal to achieve, trying a different procedure when one fails. The approach was later extended with AP [7], integrating the planner PRODIGY [92] producing RAP procedures.

PRS [42] is an action refinement and monitoring system. As in RAP, procedures specify skills to achieve goals or to react to particular events and observations. The system commits to goals and tries alternative skills when one fails. PRS relies on a database describing the world. It allows concurrent procedure execution and multi-threading. Some planning capabilities have been added to PRS [19] to anticipate paths leading to execution failure. PRS is used on various robotics platforms to trigger commands, e.g., through GenoM functional modules services [43].

Cypress [94] results from merging the planner Sipe with PRS. It uses a unified representation for planning operators and PRS skills, which was extended into the Continuous Planning and Execution Framework (CPEF) [75]. CPEF includes several components for managing and repairing plans. The system has been deployed for military mission planning and execution.

TCA [88] was initially developed to handle concurrent planning and execution. It provides a hierarchical tasks decomposition framework, with explicit temporal constraints to allow tasks synchronization. Planning is based on task decomposition. It is mostly focused on geometrical and motion planning (e.g., gait planning, footfall planning for the Ambler robot). The *Task Definition Language* (TDL) [89] 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 [4] illustrates another approach which uses transformation rules to modify hand written plans expressed in the *Reactive Plan Language* (RPL). Unlike the above systems, it explores a plan space, transforming the initial RPL relying on simulation and probabilities of possible outcomes. It replaces the currently executed plan on the fly if another one more adapted to the current situation is found. This approach evolved toward *Structured Reactive Controllers* (SRC) and *Structure Reactive Plan* (SRP) [3], but still retains the XFRM technique to perform planning using transformation rules on SRP. It has been deployed on a number of service robots at Technical University of

Munich.

Most procedure-based approaches focus on the Refinement and Instantiation/Propagation functions of an acting system. XFRM proposes a form of plan repair in plan space taking into account the probabilities of outcomes, while TDL provides some synchronization mechanism between skills and commands. All skills used by these systems are hand written, sometimes in a formalism shared with the planner (e.g., in Cypress and TCA), but without consistency checking. The hand written skills map to the robot commands, except for XFRM where some can be transformed online.

4.2 Automata–based approaches

It seems quite natural to express an abstract action as a program whose I/O are the sensory-motor signals and commands. PLEXIL, a language for the execution of plans, illustrates such a representation where the user specifies node as computational abstraction [93]. It has been developed for space applications and used with several planners such as CASPER, but it remains fairly generic and flexible. A node can monitor events, execute commands or assign values to variables. It may refer hierarchically to a list of lower level nodes. Similarly to TDL, PLEXIL execution of nodes can be controlled by a number of constraints (start, end), guards (invariant) and conditions. PLEXIL remains very focused on the execution part. But of the generated plan, it does not share knowledge with the planner.

SMACH, the ROS execution system, offers an automata-based approach [6]. The user provides a set of hierarchical automata whose nodes corresponds to components of the robot and the particular state they are in. The global state of the robot corresponds to the joint state of all components. ROS actions, services and topics (i.e. monitoring of state variables) are associated to automata states, and according to their value, the execution proceeds to the next appropriate state. An interesting property of Automata-based approaches is that the Acting component knows exactly in which state the execution is, which eases the deployment of the monitoring function.

Automata-based approaches focus on the coordination function. They can also be used for refinement and instantiation/propagation. Models are hand-written. However, the underlying formalism permits possibly a validation function with automata checking tools.

4.3 Logic–Based approaches

A few systems try to overcome the tedious engineering bottleneck of detailed hand specifications 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'll come back in section 5) and the situation calculus approach. The latter is exemplified in GOLEX [37], an execution system

for the GOLOG planner.

In GOLOG and GOLEX the user specify respectively planning and acting knowledge in the situation calculus representation. GOLEX provides Prolog "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 planning/execution interleaving.

The Logic-based approaches provides refinement and instantiation/propagation functions. But their main focus is on the logical specification of the skills, and the possibility to validate and verify their models. TAL (see section 5) offers also a Time management handling.

4.4 CSP–based approaches

Most robotics applications require explicit time to handle durative actions, concurrency and synchronization with exogenous events and other robots, and with absolute time. Several approaches manage explicit time representations by extending state-based representations with durative actions (e.g., RPG, LPG, LAMA, TGP, VHPOP, Crickey). A few of them can manage concurrency and, in the case of COLIN [15], even linear continuous change. However, temporal planners that rely on *time-lines*, i.e., partially specified evolution of state variables over time, are more expressive and flexible in the integration of planning and acting than the standard extension of state-based planners. Their representation ingredients are:

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

The initial values, expected events and goals are expressed in this representation as an *unexplained* trajectory, i.e., some required state-variable changes have to be accounted for by the planner through actions. These are instances of operators whose preconditions and effects are similarly expressed by time-lines and constraints.

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 flaw but still contain non instantiated temporal and atemporal variables. This *least commitment* approach has several advantages permitting to adapt

the acting system to the contingencies of the execution context.

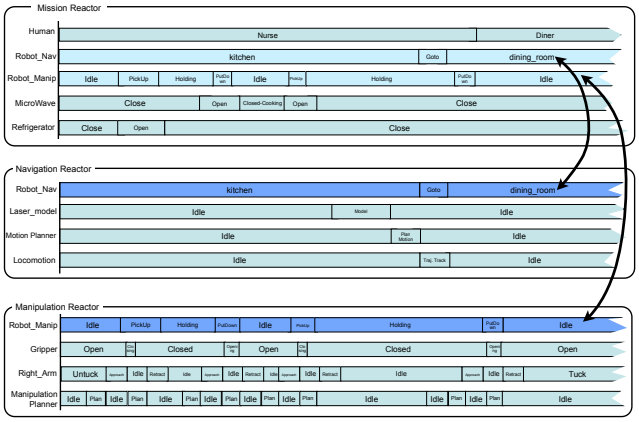

Figure 3: Example of an IDEA/T-ReX Plan built within three reactors (Mission reactor, Navigation reactor, Manipulation reactor). Note the timelines Robot_Nav and Robot_Manip shared between reactors.

The acting system proceeds like the planner by propagating execution constraints, including for observable but non controllable variables (e.g., ending time of actions). As for any CSP, consistency does not guaranty that all possible variable values are compatible. Hence the system keeps checking the consistency of the remaining plan, propagating new constraints and triggering a plan repair when needed. Special attention has to be paid to observed values of non controllable variables, depending on the requirements of strong, weak or dynamic controllability [72, 16].

IxTeT [34] is an early temporal planner along this approach that was later extended with execution capabilities [62]. Planning and acting share the partial plan structure which is dynamically produced during planning phase, and executed/repaired during the execution phase. An execution failure is seen as a new flaw in the partial plan space. The repair process minimizes as much as possible the consequences of a failure on the rest of the plan. Repair requires invalidating part of the current plan and/or relaxing some constraints. This is done by memorizing for each causal link the reason why it has been inserted (with or without a task). Causal links associated to tasks are not removed from the flawed plan. If the repair does not succeed in some allocated time, the current plan is discarded and the planning is restarted from the current situation. To fill the gap with robot commands and perceptions, PRS is used jointly with IxTeT for refinement and skills execution.

PS is another early timeline-based planning and acting system. As a component of the New Millennium Remote Agent [44, 77, 74], it controlled the Deep Space One (DS1) probe for a few days. Various components were used on DS1, PS and EXEC being the one of interest for this section (FDIR will be presented in section 5). EXEC uses a procedure-based approach, as presented in

section 4.1, without execution constraint propagation and reasoning on the current plan. In addition to its impressive application success in DS1, this system inspired the development of two interesting approaches: IDEA (then T-ReX) and RMPL.

IDEA [73] 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 into a hierarchy of *reactors*, each being in charge of a particular deliberation function: e.g. mission planning, robot navigation, robot manipulation, payload management, etc; each has its own set of timelines, planning horizon, and a computing time quantum. Reactors use the Europa planner to perform the constraint-based temporal planning. Two reactors may share timelines, accessed and modified by both, possibly with priorities. The timelines sharing mechanism allows the propagation of the planning results down to commands and similarly the integration from precepts to observations. For example, in Figure 3 the timeline robot_nav in the *mission* reactor will specify on a shared timeline with the *navigation* reactor the sequence of locations the robot must reach. From the *mission* reactor, the timeline will be seen as an "execution" one, while for the *navigation* reactor it is a "goal".

In principle the hierarchy of reactors should be able to express a continuum from planning operators down to commands. However, in practice, the shortest computing time quantum that could be achieved was in 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 reactors proved to be quite complex and tedious.

IDEA has been experimented with on several platforms such as the K9 and Gromit rovers [26]. It led to the development, along a similar approach, of a system called T-ReX, deployed at MBARI for controlling UAVs [81]. T-ReX simplifies some of IDEA too ambitious goals. For example in T-ReX, reactors are organized in such a way that constraint propagation is guaranteed to reach a fixed point (which was not the case in IDEA). T-ReX also tries to be planner independent and has been used jointly with APSI [30], yet most implementations use Europa [28].

RMPL (for Reactive Model-based Programming Language) [41], another spinoff of the DS1 experiment, proposes a common representation for planning, acting and monitoring. It combines a system model with a control model (Figure 4). The former uses hierarchical 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, as in Esterel [18]). Moreover, RMPL programs are transformed into Temporal Plan Networks (TPN) [95]. The result of each RMPL program is a partial temporal plan which is analyzed by removing flaws and transformed for execution taking into account online temporal flexibility. Altogether, RMPL offers an interesting and original integration of state-based models, procedural control and temporal reasoning used in satellite control applications.

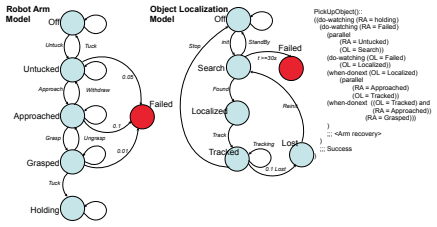

Figure 4: Example of an RMPL automata (system model on the left) and program (control model on the right). Note the non nominal outcomes (with probabilities) of actions in the system model, and the coordination operators in the control model.

CSP approaches are very convenient for handling time. They provide refinement, instantiation and, for some of them, plan repair (IxTeT, IDEA, T-ReX and RMPL). They also rely on hand written models of skills which are handled by various CSP algorithms (STN, constraints filtering). RMPL manages also nondeterminism by modeling non nominal transitions of the system.

		Functions				
Approaches	**Systems**	***Refinement***	***Instantiation***	***Time handling***	***Nondeterminism***	***Repair***
Procedure	RAP	X	X			
	PRS	X	X			
	Cypress/CPEF	X	X			
	TCA/TDL	X	X	X		
	XFRM/RPL/SRP	X	X		X	X
Automata Graph	PLEXIL	X	X	X		
	SMACH	X	X	X		
Logic	Golex	X	X			
CSP	IxTeT	X	X	X		X
	RMPL	X	X	X	X	X
	IDEA/T-ReX	X	X	X		X
	Casper	X	X	X		X
Stochastic	MCP				X	
	Pearl				X	
	Robel				X	
	Ressac				X	

Table 1: Illustrative examples of acting approaches and functions.

4.5 Stochastic–based approaches

The classical framework of Markov Decisional Processes (MDPs) offers an appealing approach for integrating planning and acting. It naturally handles probabilistic effects and it provides policies, i.e., universal plans, defined everywhere. The execution of a policy is a very simple loop: *(i)* observe current state, then *(ii)* apply corresponding action. It can even be extended, in principle, to partially observable systems (POMDP), as illustrated in the Pearl system of [79]. That framework works fine as long as the state space, together with its cost and probability parameters, can be entirely acquired and explicited, and,

for POMDPs, remains of small size.[4] 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, most of these problems are usually focused on reaching a goal from some initial state s_0. Factorized and hierarchical representations of MDPs [9], together with heuristic search algorithms for Stochastic Shortest Path (SSP) problems [65], offer a promising perspective for using effectively stochastic representations in deliberate action.

SSP problems focus on *partial policies*, *closed* for the initial state s_0 (i.e., defined on all states reachable from s_0), terminating at goal states. They generalize to And/Or graphs classical path search problems. For very large implicit search spaces, based on sparse models (few applicable actions per state, few nondeterministic effects per applicable action, including deterministic actions), a significant scaling up with respect to classical dynamic programming methods can be achieved with heuristics and sampling techniques [65].

Most heuristic search algorithms for SSPs are based on a two steps Find&Revise general framework: *(i)* Find an unsolved state s in the successors of s_0 with current policy, and *(ii)* Revise the estimated value of s along its current best action (with the so-called Bellman update). A state s is solved when the best (or a good) goal reaching policy from s has already been found. That framework can be instantiated in different ways, e.g.,

- with a best-first search, as in AO*, LAO* and their extensions (ILAO*, BLAO*, RLAO*, etc.)
- with a depth-first iterative deepening search, as in LDFS
- with a random walk along current greedy policy, as in RTDP, LRTDP and their extensions (BRTDP, FRTDP, SRTDP, etc.)

These algorithms assume an SSP problem with a *proper policy* closed for s_0 (i.e., one that reaches a goal state from s_0 with probability 1) where every improper policy has infinite cost. A generalization relaxes this last assumption and allows to seek a policy that maximizes the probability of reaching a goal state, a very useful and desirable criteria [52]. Other issues, such as dead-ends (states from which its not possible to reach a goal) have to be taken care of, in particular in critical domains [50].

Heuristic search algorithms in SSPs are more scalable than dynamic programming techniques for MDP planning, but they still cannot address large domains, with hundreds of state variables, unless these domains are carefully engineered and decomposed. Even a solution policy for such problems can be of a size so large as to make its enumeration and memorization challenging to current techniques. However, such a solution contains many states of very low probability that would almost never be visited. Various sampling and approximation techniques offer promising alternatives to further scale up probabilistic planning.

Among these approaches, determinization techniques transform each nondeterministic actions into several deterministic ones (the most likely or all pos-

[4] A POMDP is an MDP on the belief space, whose size is exponential in that of the state space. The latter is already of size k^n, for a domain with n state variables.

sible ones), then it plans deterministically with these actions, online and/or offline. For example, the RFF planner [90] generates an initial deterministic plan, then it considers a fringe state along a non-deterministic branch of that plan: if the probability to reach that state is above a threshold, it extends the plan with a deterministic path to a goal or to an already solved state.

Similar ideas are developed in sampling approaches. Among their advantages is the capability to work without a priori estimates of the probability distributions of the domain, as long as the sampling is drawn from these same distributions. Bounds on the approximation quality and the complexity of the search have been obtained, with good results on various extensions of algorithms such as LRTDP and UCT , e.g. [49, 11, 51].

Although MDPs are often used in robotics at the sensory motor level, in particular within reinforcement learning approaches, SSP techniques are not as widely disseminated at the deliberative planning and acting level. Contributions are mostly on navigation problems, e.g., the RESSAC system [91]. On sparsely nondeterministic domains where most actions are deterministic but of a few are probabilistic, the approach called MCP [64] reduces with deterministic planning a large problem into a compressed MDP. It has tested on a simulated multi-robot navigation problem.

Finally, let us mention promising heterogeneous approaches where task planning is deterministic and SSP techniques are used for the choice of the best skill refining an action, given the current context. An illustration is given by the ROBEL system [71] with a receding horizon control.

5 Monitoring

The monitoring function is in charge of *(i)* detecting discrepancies between predictions and observations, *(ii)* classifying these discrepancies, and *(iii)* recovering from them. Monitoring has at least to monitor the planner's predictions supporting the current plan. It may have also to monitor predictions made when refining plan steps into skills and commands, as well as to monitor conditions relevant for the current mission that are left implicit in planning and refinement steps. The latter are, for example, how calibrated are the robot's sensors, or how charged are its batteries.

Although monitoring functions are clearly distinct from action refinement and control functions, in many cases the two are implemented by the same process with a single representation. For example, the early Planex [25] performs a very simple monitoring through the iterated computation of the current active kernel of a triangle table. In most procedure-based systems there are PRS, RAP, ACT or TCA constructs that handle some monitoring functions. However, diagnosis and recovery functions in such systems are usually limited and ad hoc.

Diagnosis and recovery are critical in applications like the DS1 probe, for which FDIR, a comprehensive monitoring system, has been developed [74]. 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 failure states of that component. Edges are either *commands* or exogenous transition *failures*. 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 allowing for concurrent transitions compatible with constraints and preconditions. The entire model is compiled into a temporal propositional logic formula which is queried through a solver. Two query modes are used: *(i) diagnosis, i.e.*, find most likely transitions consistent with the observations, and *(ii) recovery, i.e.*, find minimal cost commands that restore the system into a nominal state. This approach has been demonstrated as being effective for a spacecraft. However, it is quite specific to cases where monitoring can be focused on the robot itself, not on the environment, and where reliability is a critical design issue addressed through redundant components permitting complex diagnosis and allowing for recovery actions. It can be qualified as a robust *proprioceptive monitoring* approach. It is unclear how it could handle environment discrepancies, *e.g.,* a service robot failing to open a door.

Other robotics monitoring systems are surveyed in [78] and characterized into three classes: analytical approaches, data-driven approaches and knowledge-based approaches. The former rely on planning and acting models, such as those mentioned above, but also control theory models and filtering techniques for low-level action monitoring. Data-driven approaches rely on statistical clustering methods for analyzing training data of normal and failures cases, and pattern recognition techniques for diagnosis. Knowledge-based approaches exploit specific knowledge in different representations (rules, chronicles, neural nets, etc.), which is given or acquired for the purpose of monitoring and diagnosis. This classification of almost 90 different contributions to Monitoring in robotics is inspired from the field of industrial control, where Monitoring is a well studied issue. However, the relationship between Monitoring, Planning and Acting was not a major concern in the surveyed contributions.

That relationship is explored in [29] on the basis of *plan invariants*. Several authors have synthesized state-reachability conditions, called invariants, from the usual planning domain specifications. Invariants permit a focus and a speed-up of planning algorithms, *e.g.*, [84, 5]. Going further, [29] proposes *extended planning problems*, where the specifications of planning operators are augmented by logical formula stating invariant conditions that have to hold during the execution of a plan. Indeed, planning operators and extended invariants are two distinct knowledge sources that have to be modeled and specified distinctly. These extended invariants are used to monitor the execution of a plan. They allow to detect infeasible actions earlier then their planned execution, or violated effects of action after their successful achievement. Furthermore, extended invariants allow to monitor effects of exogenous events and other conditions not influenced by the robot. However, this approach assumes complete sensing and perfect observation function. No empirical evaluation has been reported.

Along the same line, the approach of [24] has been tested on a simple office delivery robot. It relies on a logic-based representation of a dynamic envi-

ronment using the fluent calculus [86]. Actions are described by *normal* and *abnormal* preconditions. The former are the usual preconditions. The latter are assumed away by the planner as default; they are used as a possible explanation by the monitor in case of failure. E.g., delivery of an object to a person may fail with abnormal preconditions of the object being lost or the person not being traceable. Similarly, abnormal effects are specified. Discrepancies between expectations and observations are handled by a prioritized non-monotonic default logic, which generates explanations ranked using relative likelihood. That system handles incomplete world model and observation updates performed either 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 others authors in different settings. The interesting approach of [8] uses domain knowledge expressed in description logic to derive expectations of the effects of actions in a plan to be monitored during execution. An interesting variant is illustrated in [60] for a hybrid architecture, combining a behavior-based reactive control with model-based deliberation capabilities. At each cycle, concurrent active behaviors are combined into low-level controls. At a higher level, properties of the robot behaviors are modeled using Linear Temporal Logic (LTL). LTL formula express correctness statements, execution progress conditions, as well as goals. A trace of the robot execution, observed or predicted at planning time, is incrementally checked for satisfied and violated LTL formula. A *delayed formula progression* technique evaluates at each state the set of pending formula. It returns the set of formula that has to be satisfied by any remaining trace. The same technique is used both for Planning (with additional operator models and some search mechanism) and for Monitoring. The approach has been tested on indoor navigation tasks with robots running the Saphira architecture [54].

A very comprehensive and coherent integration of Monitoring to Planning and Acting is illustrated in the approach used in the Witas project [21]. That system demonstrates a complex Planning, Acting and Monitoring architecture embedded on autonomous UAVs. It has been demonstrated in surveillance and rescue missions. Planning relies of TALplanner [58], a forward chaining planner using the *Temporal Action Logics*(TAL) formalism for specifying planning operators and domain knowledge. Formal specifications of global constraints and dependencies, as well as of operator models and search recommendations, are used by the planner to control and prune the search. These specifications are also used to automatically generate *monitoring formula* from the model of each operator, and from the complete plan, *e.g.*, constraints on the persistence of causal links. This automated synthesis of monitoring formula is not systematic but rather selective, on the basis of hand programmed conditions of what needs to be monitored and what doesn't. In addition to the planning domain knowledge, extra monitoring formula are also specified 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 on-line, at the right moment, using formula progression techniques. When actions do not achieve their desired results, or when some other conditions fail, a recovery through a plan repair phase is triggered. Acting is performed by *Task Procedures*, which provide some level of action refinement through classical concurrent procedural execution, down to elementary commands. Altogether, this system proposes a coherent continuum from Planning to Acting and Monitoring. The only component which does not seem to rely on formal specifications is the Acting function which uses hand written Task Procedures. However the lack of flexible action refinement is compensated for by specifying planning operators (and hence plan steps) at a low-level of granularity. For example, there are five different *fly* operators in the UAV domain corresponding to different contexts, specifying context-specific control and monitoring conditions, and being mapped to different Task Procedures.

6 Perceiving

Situated deliberation relies on data reflecting the current state of the world. Beyond sensing, perceiving combines bottom-up processes from sensors to interpreted data, with top-down focus of attention, search and planning for information gathering actions. Perceiving is performed at:

- the *signal level*, e.g., signals needed in control loops ,
- the *state level*: features of the environment and the robot and their link to facts and relations characterizing the state of the world, and
- the *history level*, i.e., sequences or trajectories of events, actions and situations relevant for the robot's mission.

The signal level is usually dealt with through models and techniques of control theory. Visual servoing approaches [13] for tracking or handling objects and moving targets offer a good example of mature techniques that can be considered as tightly integrated into the basic robot functions. Similarly for simultaneous localization and mapping techniques, a very active and well advanced field in robotics, to which numerous publications have been devoted, e.g., [2, 69]. These geometric and probabilistic techniques, enriched with topological and semantic data, as for example in [56, 57, 53], may involve deliberation and can be quite effective.

But of the above areas, methods for designing perceiving functions remain today a limiting factor in autonomous robotics, a hard and challenging issue to which surprisingly not enough efforts have been devoted. The building blocks for such a function can to be taken from the fields of signal processing, pattern recognition and image analysis, which offer a long history of rich developments. However, the integration of these techniques within the requirements of autonomy and deliberation remains a bottleneck.

The *anchoring problem* provides an excellent illustration of the complexity of integrating pattern recognition methods with autonomous deliberate action. As defined in [17], 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 on objects through symbolic attributes. It is essential that the symbolic description and the sensing data agree about the objects they are referring to. 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 an internal link, called an *anchor*, between the perceptual system and the symbol system, together with a signature that gives estimate of some of the attributes of the object it refers to. The anchor is based on a model that relates relations and attributes to perceptual features and their possible values.

Establishing an anchor corresponds to a pattern recognition problem, with the challenges of handling uncertainty in sensor data and ambiguity in models, dealt with for example through maintaining multiple hypotheses. Ambiguous anchors are handled in [47] as a planning problem in a space of belief states, where actions have causal effects that change object properties, and observation effects that partition a belief state into several new hypotheses. There is also the issue of which anchors to establish, when and how, in a bottom-up or a top-down process. Anchors in principle are needed for all objects relevant to the robot mission. These objects can only be defined by intension (not extensively), in a context-dependent way. There is also the issue of tracking anchors, *i.e.*, taking into account objects 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 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 of its behavior.

The dynamics of the environment is a strong source of complexity, e.g., as we just saw in the anchor tracking and re-acquiring problems. This dynamics is itself what needs to interpreted for the history level: what an observed sequence of changes means, what can be predicted next from past evolutions. In many aspects, research at this history level is more recent. It relates to acting in and understanding environments with rich semantics, in particular involving human and man–robot interactions, e.g., in applications such as robot programming by demonstration [1] or video surveillance [40, 31].

The survey of [55] covers an extensive list of contributions to action and plan recognition. These are focused on *(i)* human action recognition, *(ii)* general activity recognition, and *(iii)* plan recognition level. The understanding is that the former two sets of processing provide input to the latter. Most surveyed approaches draw from two sources of techniques:

- *Signal processing*: Kalman and other filtering techniques, Markov Chains, Hidden Markov Models. These techniques have been successfully used in particular for movement tracking and gesture recognition[97, 67].
- *Plan recognition*: deterministic [48, 83] or probabilistic [32] planning techniques, as well as parsing techniques [82].

Most plan recognition approaches assume to get as input a sequence of symbolic actions. This assumption is reasonable for story understanding and document processing applications, but it does not hold in robotics. Usually actions are sensed only through their effects on the environment.

The *Chronicle recognition* techniques [22, 33] are very relevant at the history level of the Perceiving function. A chronicle recognition system is able to survey a stream of observed events and recognize, on the fly, instances of modeled chronicles that match this stream. A chronicle is a model for a collection of possible scenarios. It describes patterns of observed events, i.e., change in the value of state variables, persistence assertions, non occurrence assertions and temporal constraints between these assertions. A ground instance of a chronicle can be formalized as a nondeterministic timed automata. Chronicles are similar to temporal planning operators. 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 as time advances. Recent development have added hierarchization and focus on rare events with extended performances [23].

Very few systems have been proposed for designing and implementing a complete Perceiving function, integrating the three levels mentioned earlier of signal, state and history views. DyKnow [39] stands as a clear exception, noteworthy by its comprehensive and coherent approach. This system addresses several 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; it manages uncertainty, reasons on explicit models of its content and is able to dynamically reconfigure its functions.

These challenging 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 time-stamped samples representing observations or estimations of the value of a feature. It is associated with a formally specified *policy* giving requirements on its content such as: frequency of updates, 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 different types, such as *primitive processes*, that are directly connected to sensors and databases; *refinement processes*, that subscribe input streams and provide as output more combined features, e.g., a signal filter or a position estimator fusing several raw sensing sources and filtered data; or *configuration processes* that allow to reconfigure 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 Knowledge Processing Language (KPL) to specify processes, streams and corresponding policies. KPL allows to refer to objects, features, streams, processes, and time, together with their domains, constraints and relationships in the processing network. Formal specifications in KPL defines 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 to describe and enforce streams policies. They also support a number of essential functions, e.g., synchronize states from separate unsynchronized streams; evaluate incrementally temporal logic formulas over states; recognize objects and build up anchors to classify and update interpretation as new information becomes available; or follow histories of spatio-temporal events and recognize occurrences of specified chronicle models.

DyKnow has been integrated to the TALplanner system [21] discussed earlier. This system is 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.

7 Goal reasoning

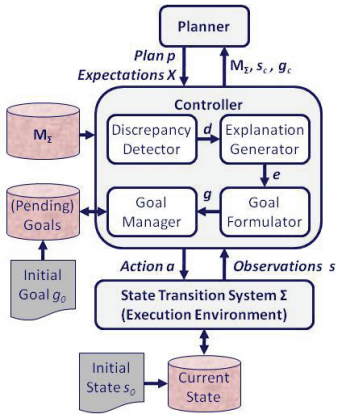

Figure 5: GDA Model with its different components, from [68].

Goal reasoning is mostly concerned with the management of high-level goals and the global mission. Its main role is to manage the set of objectives the system wants to achieve, maintain or supervise. It may react to new goals given by the user or to goal failure reported acting and monitoring. In several implementations, this function is embedded in the planning or acting functions. It clearly shares similarities with the monitoring function. Still, Goal Reasoning is not akin to planning as it does not really produce plan, but merely establish new goals and manage existing one which are then passed to the planner. Similarly to monitoring, it continuously checks unexpected events or situations. These are analyzed to assess current goals and possibly establish new goals.

Some systems have a dedicated component to perform this high-level function. For example, Goal Driven Autonomy (GDA) approaches model and reason on various and sometime conflicting goals an agent may have to consider. GDA reasoning focus on goal generation and management. In [68], the authors instantiate the GDA model in the ARTUE agent which appropriately responds to unexpected events in complex simulations and games environment. As shown on figure 5, their system includes a classical planner; when it executes a plan, it detects discrepancy (Discrepancy Detector), generates an explanation, may produce a new goal (Goal Formulator) and finally manages the goals currently under consideration by the system. The Goal Manager can use different approaches to decide which goal to keep (e.g., using decision theory to balance conflicting goals).

Similarly, in [80] the authors point out that planning should be considered from a broader point of view and not limited to the sole activity of generating an abstract plan with restrictive assumptions introduced to scope the field and make the problem more tractable. They propose the Plan Management Agent (PMA) which, beyond plan generation, provides extra plan reasoning capabilities. The resulting PMA system heavily relies on temporal and causal reasoning, and is able to plan with partial commitments, allowing to further refine a plan when needed.

Goal reasoning has been deployed in a number of real experiments. Notably in the DS1 New Millenium Remote Agent experiment [74] and in the CPEF framework [75]. Yet, overall, the goal reasoning function is not often developed. It is nevertheless needed for complex and large systems managing various long term objectives while taking dynamically into account new events which may trigger new goals.

8 Integration and Architectures

Beyond the integration of various devices (mechanical, electrical, electronical, etc), robots are complex systems including multiple sensors, actuators and information processing modules. They embed online processing, with various real time requirement, from low-level servo loops up to deliberation functions which confer the necessary autonomy and robustness for the robot to face the variability of tasks and environment. The software integration of all these components must rely on an architecture and supporting tools which specify how theses components communicate, share resources and CPUs, and how they are implemented on the host computer(s) and operating systems.

Various architectures have been proposed to tackle this task, among which the following:

- *Reactive architectures*, e.g. the subsumption architecture [10], are composed of modules which close the loop between inputs (e.g. sensors) and outputs (e.g. effectors) with an internal automata. These modules can be hierarchically organized and can inhibit other modules or weight on their activity. They do not rely on any particular model of the world or plans to achieve and do not

support any explicit deliberative activities. Nevertheless, there are a number of work, e.g. [59], which rely on them to implement deliberative functions.

- *Hierarchical architectures* are probably the most widely used in robotics [43, 76, 20]. They propose an organization of the software along layers (two or three) with different temporal requirements and abstraction levels. Often, there is a functional layer containing the low-level sensors–effectors–processing modules, and a decision layer containing some of the deliberation functions presented here (e.g. planning, acting, monitoring, etc).
- *Teleo-reactive architectures* [26, 66] are more recent. They propose an integrated planning–acting paradigm which is implemented at different levels, from deliberation down to reactive functions, using different planning–acting horizons and time quantum. Each planner–actor is responsible for ensuring the consistency of a constraint network (temporal and atemporal) whose state variables can be shared with other planners–actors to provide a communication mechanism.

Beyond architecture paradigms, it is interesting to note that some robotics systems have achieved an impressive level of integration of numerous deliberation functions on real platforms. The Linkoping UAV project [20] provides planning, acting, perception, monitoring with formal representations all over these components. The NMRA on the DS1 probe [74] also proposed planning, acting, and FDIR onboard. IDEA and T-ReX, providing planning and acting have been used respectively on a robot [26] and an AUV [66].

9 Conclusion

Autonomous robots facing a variety of open environments and a diversity of tasks cannot rely on the decision making capabilities of a human designer or teleoperator. To achieve their missions, they have to exhibit complex reasoning capabilities required to understand their environment and current context, and to act deliberately, in a purposeful, intentional manner. In this paper, we have referred to these reasoning capabilities as deliberation functions, closely interconnected within a complex architecture. We have presented an overview of the state of the art for some of them.

For the purpose of this overview, we found it clarifying to distinguish these functions with respect to their main role and computational requirements: the *perceiving, goal reasoning, planning, acting* and *monitoring* functions. But let us insist again: the border line between them is not crisp; the rational for their implementation 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 signals and commands, to the most "offline" outer loop.

Consider for example the relationship between planning and acting. We argued that acting cannot be reduced to "execution control", that is the triggering of commands mapped to planned actions. There is a need for significant deliberation to take place between what is planned and the commands achieving it (Fig.

2). This acting deliberation may even rely on the same or on different planning techniques as those of the planner, but it has to take into account different state spaces, action spaces and event spaces than those of the planner. However, if we insisted to distinguish these two levels, there is no reason to believe that just two 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 convenient and elegant to address this hierarchy within a homogeneous approach, e.g., HTN or AHP. But we strongly suspect that conflicting requirements, e.g., for handling uncertainty and domain specific representations, favor a variety of representations and approaches.

Many other open issues, briefly referred to in this paper, give rise to numerous scientific challenges. The relationship from sensing and acting to perceiving is clearly one of these bottleneck problems to which more investigation efforts need to be devoted. Acting in an open world requires going from anchoring to symbol grounding, from object recognition to categorization. A development perspective is to make robots query when needed and benefit from the growing wealth of knowledge available over the web, within ontologies of textual and symbolic relations, as well as of images, graphical and geometric knowledge.

Deliberation functions involve several other open issues that we have not discussed in this overview, among which the noteworthy problems of:

- *metareasoning*: trading off deliberation time for acting time, given how critical and/or urgent are the context and tasks at hand;
- *interaction and social behavior* that impact all functions discussed here, from the perceiving requirements of a multi-modal dialogue, to the planning and acting at the levels of task sharing and plan understanding for multi-robots and man-robot interaction;
- *learning* which is the only hope for building the models required by deliberation functions and which has a strong impact on the architecture that would permit to integrate these functions and allow them to adapt to tasks and environments the robot is facing.

We believe that the AI–Robotics synergy is becoming richer and more complex, and it remains today as fruitful for both fields as it used to be in their early beginning. We do hope that this overview will attract more practitioners to the challenging problems of their intersection.

Acknowledgements

We thank the editors of this special issue and the reviewers for their highly valuable feedback.

References

[1] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[2] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (SLAM): part II. *IEEE Robotics and Automation Magazine*, 13(3):108 – 117, 2006.

[3] M. Beetz. Structured reactive controllers: controlling robots that perform everyday activity. In *Proceedings of the annual conference on Autonomous Agents*, pages 228–235. ACM, 1999.

[4] M. Beetz and D. McDermott. Improving Robot Plans During Their Execution. In *Proc. AIPS*, 1994.

[5] S. Bernardini and D. Smith. Finding mutual exclusion invariants in temporal planning domains. In *Seventh International Workshop on Planning and Scheduling for Space (IWPSS)*, 2011.

[6] J. Bohren, R. Rusu, E. Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mosenlechner, W. Meeussen, and S. Holzer. Towards autonomous robotic butlers: Lessons learned with the PR2. In *Proc. ICRA*, pages 5568–5575, 2011.

[7] R. Bonasso, R. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2/3):237–256, April 1997.

[8] A. Bouguerra, L. Karlsson, and A. Saffiotti. Semantic Knowledge-Based Execution Monitoring for Mobile Robots. In *Proc. ICRA*, pages 3693–3698, 2007.

[9] C. Boutilier, T. Dean, and S. Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of AI Research*, 11:1–94, May 1999.

[10] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[11] L. Busoniu, R. Munos, B. De Schutter, and R. Babuska. Optimistic planning for sparsely stochastic systems. *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning*, pages 48–55, 2011.

[12] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, 28(1):104–126, 2009.

[13] F. Chaumette and S. Hutchinson. Visual servo control, part ii: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118, 2007.

[14] F. Chaumette and S. Hutchinson. Visual servoing and visual tracking. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, pages 563–583. Springer, 2008.

[15] A. J. Coles, A. Coles, M. Fox, and D. Long. COLIN: Planning with Continuous Linear Numeric Change. *Journal of AI Research*, 2012.

[16] P. Conrad, J. Shah, and B. Williams. Flexible execution of plans with choice. In *Proceedings of ICAPS*, 2009.

[17] S. Coradeschi and A. Saffiotti. An introduction to the anchoring problem. *Robotics and Autonomous Systems*, 43(2-3):85–96, 2003.

[18] E. Coste-Maniere, B. Espiau, and E. Rutten. A task-level robot programming language and its reactive execution. In *Proc. ICRA*, 1992.

[19] O. Despouys and F. Ingrand. Propice-Plan: Toward a Unified Framework for Planning and Execution. In *European Workshop on Planning*, 1999.

[20] P. Doherty, J. Kvarnström, and F. Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19(3), 2009.

[21] P. Doherty, J. Kvarnström, and F. Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19(3):332–377, 2009.

[22] C. Dousson, P. Gaborit, and M. Ghallab. Situation recognition: Representation and algorithms. *Proc. IJCAI*, 13:166–166, 1993.

[23] C. Dousson and P. Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *Proc. IJCAI*, pages 324–329, 2007.

[24] M. Fichtner, A. Großmann, and M. Thielscher. Intelligent execution monitoring in dynamic environments. *Fundamenta Informaticae*, 57(2-4):371–392, 2003.

[25] R. Fikes. Monitored Execution of Robot Plans Produced by STRIPS. In *IFIP Congress*, Ljubljana, Yugoslavia, August 1971.

[26] A. Finzi, F. Ingrand, and N. Muscettola. Model-based executive control through reactive planning for autonomous rovers. In *Proc. IROS*, volume 1, pages 879–884, 2004.

[27] R. Firby. An Investigation into Reactive Planning in Complex Domains. In *Proc. AAAI*, pages 1–5, 1987.

[28] J. Frank and A. Jónsson. Constraint-based attribute and interval planning. *Constraints*, 8(4):339–364, 2003.

[29] G. Fraser, G. Steinbauer, and F. Wotawa. Plan execution in dynamic environments. In *Innovations in Applied Artificial Intelligence*, volume 3533 of *LNCS*, pages 208–217. Springer, 2005.

[30] S. Fratini, A. Cesta, R. De Benedictis, A. Orlandini, and R. Rasconi. Apsi-based deliberation in goal oriented autonomous controllers. In *11th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2011.

[31] F. Fusier, V. Valentin, F. Brémond, M. Thonnat, M. Borg, D. Thirde, and J. Ferryman. Video understanding for complex activity recognition. *Machine Vision and Applications*, 18:167–188, 2007.

[32] C. Geib and R. Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 173:1101–1132, 2009.

[33] M. Ghallab. On chronicles: Representation, on-line recognition and learning. In *International Conference on Knowledge Representation and Reasoning*, pages 597–606, 1996.

[34] M. Ghallab and A. Mounir Alaoui. Managing efficiently temporal relations through indexed spanning trees. In *Proc. IJCAI*, pages 1297–1303, 1989.

[35] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice.* Morgann Kaufmann, October 2004.

[36] G. Giralt, R. Sobek, and R. Chatila. A multi-level planning and navigation system for a mobile robot: a first approach to HILARE. In *Proc. IJCAI*, pages 335–337, 1979.

[37] D. Hähnel, W. Burgard, and G. Lakemeyer. GOLEX—bridging the gap between logic (GOLOG) and a real robot. In *KI-98: Advances in Artificial Intelligence*, pages 165–176. Springer, 1998.

[38] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 220–229, 1998.

[39] F. Heintz, J. Kvarnström, and P. Doherty. Bridging the sense-reasoning gap: DyKnow-Stream-based middleware for knowledge processing. *Advanced Engineering Informatics*, 24(1):14–26, 2010.

[40] S. Hongeng, R. Nevatia, and F. Bremond. Video-based event recognition: activity representation and probabilistic recognition methods. *Computer Vision and Image Understanding*, 96(2):129–162, 2004.

[41] M. Ingham, R. Ragno, and B. Williams. A Reactive Model-based Programming Language for Robotic Space Explorers. In *International Symposium on Artificial Intelligence, Robotics and Automation for Space*, 2001.

[42] F. Ingrand, R. Chatilla, R. Alami, and F. Robert. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE International Conference on Robotics and Automation*, 1996.

[43] F. Ingrand, S. Lacroix, S. Lemai-Chenevier, and F. Py. Decisional Autonomy of Planetary Rovers. *Journal of Field Robotics*, 24(7):559–580, October 2007.

[44] A. Jónsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in Interplanetary Space: Theory and Practice. In *International Conference on AI Planning Systems*, 2000.

[45] L. Pack Kaelbling and T. Lozano-Perez. Hierarchical task and motion planning in the now. In *Proc. ICRA*, pages 1470–1477, 2011.

[46] O. Kanoun, J-P. Laumond, and E. Yoshida. Planning foot placements for a humanoid robot: A problem of inverse kinematics. *International Journal of Robotics Research*, 30(4):476–485, 2011.

[47] L. Karlsson, A. Bouguerra, M. Broxvall, S. Coradeschi, and A. Saffiotti. To secure an anchor – a recovery planning approach to ambiguity in perceptual anchoring. *AI Communications*, 21(1):1–14, 2008.

[48] H. Kautz and J. Allen. Generalized plan recognition. In *Proc. AAAI*, pages 32 – 37, 1986.

[49] M. Kearns, Y. Mansour, and A. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49:193–208, 2002.

[50] A. Kolobov, Mausam, and D. Weld. SixthSense: Fast and Reliable Recognition of Dead Ends in MDPs. *Proc. AAAI*, April 2010.

[51] A. Kolobov, Mausam, and D. Weld. LRTDP vs. UCT for Online Probabilistic Planning. In *Proc. AAAI*, 2012.

[52] A. Kolobov, Mausam, D. Weld, and H. Geffner. Heuristic search for generalized stochastic shortest path MDPs. *Proc. ICAPS*, 2011.

[53] K. Konolige, E. Marder-Eppstein, and B. Marthi. Navigation in hybrid metric-topological maps. In *Proc. ICRA*, 2011.

[54] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti. The saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:215–235, 1997.

[55] V. Kruger, D. Kragic, A. Ude, and C. Geib. The meaning of action: a review on action recognition and mapping. *Advanced Robotics*, 21(13):1473–1501, 2007.

[56] B. Kuipers and Y. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8(1-2):47–63, 1991.

[57] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli. Local metrical and global topological maps in the hybrid spatial semantic hierarchy. In *Proc. ICRA*, pages 4845–4851, 2004.

[58] J. Kvarnström and P. Doherty. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30(1):119–169, 2000.

[59] K. Ben Lamine and F. Kabanza. History checking of temporal fuzzy logic formulas for monitoring behavior-based mobile robots. In *12th IEEE International Conference on Tools with Artificial Intelligence, 2000. ICTAI 2000.*, pages 312–319, 2000.

[60] K. Ben Lamine and F. Kabanza. Reasoning about robot actions: A model checking approach. In M. Beetz, J. Hertzberg, M. Ghallab, and M. Pollack, editors, *Advances in Plan-Based Control of Robotic Agents*, volume 2466 of *LNCS*, pages 123–139, 2002.

[61] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[62] S. Lemai-Chenevier and F. Ingrand. Interleaving Temporal Planning and Execution in Robotics Domains. In *Proc. AAAI*, 2004.

[63] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.

[64] M. Likhachev, G. Gordon, and S. Thrun. Planning for markov decision processes with sparse stochasticity. *Advances in Neural Information Processing Systems (NIPS)*, 17, 2004.

[65] Mausam and A. Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool, July 2012.

[66] C. McGann, F. Py, K. Rajan, J. Ryan, and R. Henthorn. Adaptive Control for Autonomous Underwater Vehicles. In *Proc. AAAI*, page 6, April 2008.

[67] T. Moeslund, A. Hilton, and V. Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90–126, 2006.

[68] M. Molineaux, M. Klenk, and D. Aha. Goal-driven autonomy in a Navy strategy simulation. In *Proc. AAAI*, pages 1548–1554, 2010.

[69] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. IJCAI*, pages 1151–1156, 2003.

[70] H. Moravec. The Stanford Cart and the CMU Rover. Technical report, CMU, February 1983.

[71] B. Morisset and M. Ghallab. Learning how to combine sensory-motor functions into a robust behavior. *Artificial Intelligence*, 172(4-5):392–412, March 2008.

[72] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *Proc. IJCAI*, pages 494–502, 2001.

[73] N. Muscettola, G. Dorais, C. Fry, R. Levinson, and C. Plaunt. A Unified Approach to Model-Based Planning and Execution. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, 2000.

[74] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 103:5–47, 1998.

[75] K. Myers. CPEF: Continuous Planning and Execution Framework. *AI Magazine*, 20(4):63–69, 1999.

[76] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin. CLARAty and Challenges of Developing Interoperable Robotic Software. In *Proc. IROS*, October 2003.

[77] B. Pell, E. Gat, R. Keesing, N. Muscettola, and Ben Smith. Robust Periodic Planning and Execution for Autonomous Spacecraft. In *Proc. IJCAI*, 1997.

[78] O. Petterson. Execusion monitoring in robotics: a survey. *Robotics and Autonomous Systems*, 53:73–88, 2005.

[79] J. Pineau, M. Montemerlo, M. Pollack, N Roy, and S. Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3-4):271–281, March 2003.

[80] M. Pollack and J. Horty. There's more to life than making plans: plan management in dynamic, multiagent environments. *AI Magazine*, 20(4):71, 1999.

[81] F. Py, K. Rajan, and C. McGann. A systematic agent framework for situated autonomous systems. In *Proc. AAMAS*, pages 583–590, 2010.

[82] D. Pynadath and M. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proc. Uncertainty in Artificial Intelligence*, pages 507–514, 2000.

[83] M. Ramirez and H. Geffner. Plan recognition as planning. In *Proc. IJCAI*, pages 1778 – 1783, 2009.

[84] J. Rintanen. An iterative algorithm for synthesizing invariants. In *Proc. AAAI*, pages 806–811, 2000.

[85] C. Rosen and N. Nilsson. Application of intelligent automata to reconnaissance. Technical report, SRI, November 1966.

[86] E. Sandewall. *Features and Fluents*. Oxford university Press, 1995.

[87] T. Siméon, J-P. Laumond, J. Cortés, and A. Sahbani. Manipulation planning with probabilistic roadmaps. *International Journal of Robotics Research*, 23(7-8):729–746, 2004.

[88] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

[89] R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proc. IROS*, 1998.

[90] F. Teichteil-Königsbuch, U. Kuter, and G. Infantes. Incremental plan aggregation for generating policies in MDPs. In *Proc. AAMAS*, pages 1231–1238, 2010.

[91] F. Teichteil-Königsbuch, C. Lesire, and G. Infantes. A generic framework for anytime execution-driven planning in robotics. In *Proc. ICRA*, pages 299–304, 2011.

[92] M. Veloso and P. Rizzo. Mapping planning actions and partially-ordered plans into execution knowledge. In *Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, pages 94–97, 1998.

[93] V. Verma, T. Estlin, A. Jónsson, C. Pasareanu, R. Simmons, and K. Tso. Plan execution interchange language (PLEXIL) for executable plans and command sequences. In *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2005.

[94] D. Wilkins and K. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6):731–761, 1995.

[95] B. Williams and M. Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proc. IJCAI*, 2001.

[96] J. Wolfe, B. Marthi, and S. Russell. Combined task and motion planning for mobile manipulation. In *Proc. ICAPS*, 2010.

[97] Y. Wu and T. Huang. *Vision-Based Gesture Recognition: A Review*, volume 1739 of *LNCS*, pages 103–116. Springer-Verlag, 1999.