Software tools for the cognitive development of autonomous robots

P. Jiménez

Institut de Robòtica i Informàtica Industrial (CSIC - UPC)

Llorens i Artigas 4-6, E-08028 Barcelona, Spain

e-mail: pjimenez@iri.upc.edu

April 3, 2017

Robotic systems are evolving towards higher degrees of autonomy. This paper reviews the cognitive tools available nowadays for the fulfillment of abstract or long-term goals as well as for learning and modifying their behavior.

Contents

Introduction

Machines are powered artifacts intended at performing a given action. They can be viewed as more or less sophisticated tools, designed for executing specific work. Among all the machines developed by human mind, **robots** deserve a special place. The standard definition (ISO 8373) of an industrial robot stresses the following key aspects of its unique nature:

- Automatic control: robots deploy their activity without intervention of a human.
- Multifunctionality: robots are not constrained to perform a single, specialized action, but the same robot arm can execute as different tasks as manipulation, painting, welding, or inspection, with a simple (possibly automated) tool change.
- Reprogrammability: the trajectories and tasks developed by the robot can be easily modified by software, i.e., rewriting and/or executing a different program, without the need of readjusting the hardware.
- Continuous workspace: the robot can position itself in any point within its reach and follow arbitrary trajectories.

These features distinguish industrial robots from conventional machines. However, the degree of autonomy of such robots is heavily restricted by their environment: they are only able to deal with f incidences within a quite constrained and structured world. They are certainly able to adjust their trajectories to the actual position and orientation of arriving workpieces, or to react to failures like the absence of a piece supposed to be there or to observed defects. They can even respond to certain non-trivial sensory input like computer vision, e.g. for classification or failure detection. But to cope with the contingencies of the unstructured world outside controlled environments, some more steps have to be taken. First, sensory input becomes now an unavoidable must. The world is continuously changing, and agents like robots need to have an updated picture of the world's state in order to be able perform meaningful actions. Basic home robots like vacuum cleaners rely on very simple bug-like sensor systems based on infrared range measuring and contact detection. This suffices for the duties they are endowed with. The environment is now not as structured as in the case of industrial robots, but it has still well-defined features: a home vacuum-cleaner robot is not expected to be deployed on the streets, and a lawn-mover robot will rapidly be stuck in the savannah (if not attacked by a cheetah!). Moreover, the work they have to perform is quite simple and specific.

More advanced interaction with the surroundings like object manipulation or cooperation with humans requires a quite more sophisticated perception. Computer vision techniques are to be applied for image processing, object (or people) recognition, and scene understanding (i.e., annotating its elements and establishing their –mostly spatial– relationships). In many cases, the robot will need to *learn* what it is watching at. Learning in perception is in first instance a classification or categorization issue: when not predefined by the human through a set of salient features, classes

have to be constructed from a number of labelled examples, whose attributes have to be generalized so that in a later recognition phase the specific object perceived is correctly fit within its corresponding class. The classified object is automatically bestowed with the characteristic attributes of its class. Nowadays, with tools like the Convolutional Neural Networks (CNNs) computer vision is quite reliable in classification, even with regard to visual hindrances like poor illumination or partial occlusions. However, artificial systems are still far from human performance with respect to interpreting a scene, specially if a number of commonsense or cultural cues are involved.

Perception alone does not suffice, the robot must also be able to decide which action to perform next and to execute this action. Decision making in Robotics is obviously intended to be an autonomous process. The scope of the required cognitive load ranges from purely reactive to highly deliberative systems. These different approaches are briefly reviewed in Section "Decision making in Robotics". They provide the framework within which the cognitive processes that are needed to grant autonomy to the robotic agent take place. The term *cognitive process* has to be interpreted as referring to those algorithmic procedures that mimic higher mental processes in humans, without pretending the machine to own a mind.

Short-term decisions taken by a robot are not quite difficult to trace (i.e., to follow the process or to understand how they have been taken) or even to predict, in general. More involved are long-term decisions, as they rely on a planning process that may include a high number of planning operators (PO) that represent the individual actions, and a much higher number of possible combinations of such POs. These POs may be either provided as a fixed repertoire by the human designer or programmer, or they can be learned by the robotic agent directly from its perception channels. This said, it is clear that the two relevant cognition processes involved in decision making are learning and planning. Despite learning takes place earlier in the information processing flow than planning (we will see that this is not always true), we will examine planning first (Section "Planning") in order to get a better idea of what has actually to be learned, and then some learning paradigms will be overviewed in Section "Learning". These two sections constitute the core of this presentation, and besides providing a brief description of the corresponding processes, we will emphasize to which extent their outcomes are determined by human design or intervention. A summary and some reflections on the latter are finally provided in the concluding remarks of the paper.

Decision making in Robotics

The behavior of robots is managed by the control system, like in any automatic machine. In the case of robots, this control is heavily software-based. The computer programs that decide in any moment how the robot will respond to specific stimuli and how it will perform its duties have been designed and encoded by human programmers and users. The different control schemata can be summarized as in[1]:

1. Reactive control ("don't think, react")
2. Deliberative control ("think, then act")
3. Hybrid control ("think and act independently in parallel")
4. Behavior-based control ("think the way you act").

1 Maja Matarić, 'Learning in behavior-based multi-robot systems: Policies, models, and other agents' (2001) 2(1) Cognitive Systems Research 81 ⟨http://dx.doi.org/10.1016/S1389-0417(01)00017-1⟩ .

Purely reactive controls are also the most predictable ones -up to failures-, as any response to foreseen inputs has been previously programmed by the human, and unforeseen inputs are simply ignored. The main advantage of this control architecture is its swift response, its main drawback its lack of flexibility. Deliberative systems, on the contrary, thanks to their powerful cognitive tools -which are the subject of the rest of this contribution-, display a far more elaborated adaptability to changing conditions of the environment. Nonetheless, their response may come too late for a system which is embedded in the real world with all its threats and contingencies, as such deliberative processes are highly time- and resource-consuming. This justifies the hybrid control architecture, which provides immediate response to urgent issues (as long as they are predictable) thanks to the reactive component, whereas they are also capable of long-term adaptation to the world changes (sometimes such systems are also said as being both goal and event driven).

As for behavior-based control, it will not be tackled in this paper, but it deserves some words before proceeding to the cognitive processes. Behaviors can be thought of as small programs encoding input-output responses, operating at different levels of abstraction (without making any hierarchical structure explicit) and highly interconnected. The famous behavior-based subsumption architecture of Rodney Brooks is the paradigmatic example of this control concept, postulating that complex behaviors can *emerge* from this interconnectivity and activity of simple behaviors. Predictability goes lost as the complexity of the system grows. The idea of emergent behavior appears also in *swarm robotics*, each robotic unit being quite simple but the whole swarm being highly interconnected. Thanks to this dense communication and feedback, simple rules can make emerge complex behaviors.

As just said, we will concentrate on the cognitive processes of deliberative systems. Although they are presented independently from one another, it should be clear that reasoning constitutes in some sense the theoretical background of the other functions, and planning and learning are intimately related, as the trajectories or the symbolic actions used by the planner may be the output of the learner. Moreover, planning may be part of the learning process (at symbolic level): the planner uses the rules as learned so far to determine whether they are enough and correct to allow the accomplishment of the task. If not, the necessary modifications in the symbolic representation of actions have to be undertaken (rule refinement). Thus planning does not only provide the required instruction sequence for task fulfillment (to be translated into robot commands) but also plays an active role in the learning process.

Knowledge representation and reasoning

Reasoning occupies the highest level in an intelligent robot's control architecture. Reasoning operates on a certain kind of knowledge representation (KR), which in Robotics may belong to one of three types:

- Logic-based KR. Statements are either true or false, and knowledge about the world's state is assumed to be complete.
- Probabilistic formulations. Statements are either true or false, but their particular truth value may be unknown.
- Fuzzy logic formalism. Instead of true or false, a statement may be true up to some degree. Stated differently: while in conventional logic the truth value belongs to the set $\{0, 1\}$, in fuzzy logic it can be any value within the interval $[0, 1]$.

Next, the main features of these KR and associated reasoning types are presented.

Logic

Logic, and more specifically *First Order Predicate Logic* (FOPL) is the archetypal KR in Artificial Intelligence, where reasoning is based on the powerful *deductive* mechanism (some reasoning systems may use inductive, abductive, or other types of logic inference mechanisms as well). Under simplifying assumptions, logic formulations and its derivate action planning mechanisms (see Section "Deterministic action planning") have been used in Robotics with some success. However, Robotics intrinsically comes with two hard problems for logic formalism. The first one is that robots are physical agents embedded in a changing world, whose actions are responsible of some of these changes. In logical terms, they have changed the truth value of some facts about the world, while others remain unchanged. Determining what changes and what remains may not be a trivial issue (this is known as the *frame problem*, see also again Section "Deterministic action planning"). The second point is that sensor information may conflict with previous beliefs, i.e. state a contradictory truth value about a known fact. Mechanisms may be foreseen for providing plausible explanations and resolve such conflicts, but again, this is not trivial to resolve. Such problems render a logical formulation of robots acting in the world as undecidable in general. Practical solutions consist in considering holding periods for formulas (the notion of *situation*), or in sacrificing completeness[2].

Description logics (DL) have been consolidating their suitability for structuring semantic knowledge about the world. By inheriting some of the features of classical KR formalisms like semantic networks and frame systems, they not only provide semantic structure and consistency to a particular domain, but also some form of inference, and thus they can be seen as forming "a certain family of decidable subsets of FOPL"[3]. DLs entail two main components:

- the upper ontology or *terminological knowledge* (*TBox*), that is, the set of concepts of a particular domain and the relationships between these concepts, in particular equality and subconcept or superclass-subclass relation, the latter enabling property inheritance by creating a hierarchical taxonomy. *Concepts* are unary predicates, and concept conjunction, disjunction and negation may be used as combining operators. *Roles* are binary predicates that allow to express a semantic relation between two concepts.

- individual objects or *assertional knowledge* (*ABox*), for grounding concepts and roles.

DLs come with some reasoning basics like consistency of the concept definition, subsumption and disjointness of concepts, consistency of the *ABox* with respect to the *TBox*, concept and role instances, all of which are decidable in a DL[4]. DLs are explained in more detail in a number of works, see[5] for a good introduction. DLs are at the base of web ontology languages, like OWL, devised at developing the semantic net. As for Robotics, serious efforts have been made to derive an ontology that allows sharing and exchanging knowledge between robots about objects, tasks and environments, like KNOWROB, based on DL, which uses OWL and exploits its hierarchical structure of classes that allows inheritance[6]. This has been extended with meta-information about the data to be exchanged, algorithms that were used for creating data and requirements that are

2 Joachim Hertzberg and Raja Chatila, 'AI Reasoning Methods for Robotics' in *Springer Handbook of Robotics* (2008) ⟨http://dx.doi.org/10.1007/978-3-540-30301-5_10⟩ .

3 ibid.

4 ibid.

5 Franz Baader, Ian Horrocks, and Ulrike Sattler, 'Description Logics' in Steffen Staab and Rudi Studer (eds), *Handbook on Ontologies* (Springer Berlin Heidelberg 2004) ⟨http://dx.doi.org/10.1007/978-3-540-24750-0_1⟩ .

6 M Tenorth and M Beetz, 'KNOWROB knowledge processing for autonomous personal robots' (October 2009) ⟨http://dx.doi.org/10.1109/IROS.2009.5354602⟩ .

needed for interpreting it, aiming at robots-to-robots sharing of knowledge across a robotic World Wide Web in the ROBOEARTH project[7]. In a top-ranked international professional association like the IEEE Robotics and Automation Society, a Working Group on "Ontology for Robotics and Automation" is active since 2012 and have developed already a standard on this subject[8].

Probabilistic formulations

As embedded creatures in the world, robots rely on sensors as the main source of information about their surroundings. However, too often such information is noisy or incomplete, leading to lack of information. But almost as frequently, this does not mean an absolute ignorance about a given world state, but some kind of knowledge about the chances of the different alternatives is available in general. This can be formalized quantitatively with probabilities associated to each option, and dealt with using tools like Bayes' rule. This is an inference mechanism for computing the probability of certain event, given the priors and dependent relevant probabilities. The practical implementation of this mechanism, that allows to know the probability of a certain cause being behind the observed effect, while avoiding the huge computational cost of a naive use of this mechanism, is known as *Bayes networks* (BNs), and for systems evolving with time, *dynamic Bayesian networks* (DBNs).

Fuzzy logic

Fuzzy logic allows to reason about qualitative and approximate statements. It can be seen as a generalization of propositional logic with continuous truth values along the interval $[0, 1]$, and reformulating logical junctors to operate with such numerical values (negation as the complementary to 1, disjunction as the maximum, conjunction as the minimum, etc.). Fuzzy logic knowledge bases, containing sets of *if-then* rules that relate fuzzy values of some variables to the fuzzy values of others, can be used for inference by forward chaining. Once the fuzzy value of a given variable is computed, it can be defuzzyfied by assigning a scalar value like the central point of the corresponding interval of possible values of this variable (if such a numerical value is necessary for the application at hand).

Planning

Medium- or long-term goals require some kind of planning. Here the absolute timescale is not so relevant as the evolution of the world's state: long-term would refer, with this precision in mind, a time span in which many changes take place. This is generally associated to the fact that the robot has to concatenate a sequence of actions to achieve such a goal. Each such action modifies the world's state, either just by changing the robot's configuration, or by altering some aspects of the surroundings.

In Robotics two main types of planning have to be distinguished: motion and task planning. These two types occupy different levels as for degree of abstraction: task planning occurs at a formalistic, symbolic level, whereas motion planning takes place in a geometric mock-up of the real world. In fact, motion planning could be considered to connect task planning to the real execution of the action commands, as long as such actions involve the displacement of the robot (or of

7 M Tenorth and others, 'Representation and Exchange of Knowledge About Actions, Objects, and Environments in the RoboEarth Framework' (2013) 10(3) IEEE Transactions on Automation Science and Engineering 643 ⟨http://dx.doi.org/10.1109/TASE.2013.2244883⟩ .

8 http://standards.ieee.org/develop/wg/ORA.html

a part of it). Action specifications are generally qualitative, whereas motion guidelines are quantitative, and despite having to consider additional control issues, their translation into executable motion commands is rather straightforward.

In the next sections the main traits and variants of the two types of planning are overviewed, in order to provide a rough idea of how they work and to which extent human intervention conditions the outcome of the planner.

Motion planning

The motion planning problem is formalized as given an initial (or start) and a final (or goal) configuration of the robot, and a description of the free-space (or, in a complementary fashion, of the present obstacles), to determine a collision-free motion from the start to the goal. The notion of configuration space (C-space) is quite useful: the dimensions of such space correspond to the degrees of freedom of the robot, and therefore in such a space the robot becomes a point, and the solution path is unidimensional. The counterpart is that the planning space has now as many dimensions as the number of degrees of freedom of the robot, and the layout of such a space is not quite intuitive. C-obstacles are all the configurations resulting in a collision of the robot with surrounding objects.

Construction of the planning space

In academic toy examples, a geometrical description of the environment where planning takes place is already assumed to be provided. Other sources of existing environment descriptions are architectural plans, city maps, roadmaps, geographical maps, etc. The problem is that such maps often do not provide the level of granularity or detail required by the robot, and more importantly, they almost surely do not reflect the real layout as for the presence of obstacles. Furniture is displayed at a fixed position in an architectural plan, but chairs can be displaced and occupy unsuspected locations. A country map may not show the fences crossing a path, not to speak from all the mobile obstacles encountered in urban or rural environments, etc. Therefore, if such information is used, it has to be complemented with online observations of the environment by the robot. Aerial photographs or videos taken by a drone may provide such updated information to ground robots. Alternatively, mobile robots equipped with online cameras and/or range sensors can construct their own maps, following the techniques generically known as Simultaneous Localization and Mapping (SLAM). Robust algorithms exist nowadays for solving this task.

C-obstacles are difficult to construct explicitly, besides very simple cases. The randomized motion planning algorithms explained below, however, avoid this step and resort to efficient collision-detection algorithms only when needed.

Sensor-based motion planning

It could be the case that no geometric description of the environment is available at all. Moreover, the robot may be equipped with just quite simple onboard contact or range sensors. Even with such limitations it is possible to derive some strategies that allow the robot to find a path leading to the goal. They are known as *bug-algorithms*, because such robots really act as simple animals with limited perception. The robot is assumed to have some notion, at any location, of where the goal is, and proceeds towards it in a straightforward fashion and surrounding obstacles found on its way.

Classical motion planning

These methods operate on full descriptions of the C-space, i.e. a geometrical model of the space is available. They include **Potential functions**, **Roadmap methods**, **Exact** and **Approximate cell decompositions**. These methods really work well only for simple low-dimensional settings. Practical methods, working efficiently for real robots, rely on probabilistic approaches, and have to sacrifice completeness for efficiency. They are shown next.

Randomized motion planning

This family of methods bases its success on sampling the C-space and computing possible robot-obstacle collisions only at the sampled configurations, as well as at some points of the segments joining them. These methods are said to be probabilistic complete: if a solution path exists, the algorithm will eventually find it (the probability is higher the more samples are used). The two main families of methods are the **Multi-query planners** where the constructed roadmap can be used for different queries (i.e., finding the path between different start-goal point pairs)., the paradigmatic approach being the *probabilistic roadmap method* (PRM), whereas the **Single-query planners** consider exclusively a specific start-goal pair, constructing on the fly a tree-structure for this planning query, with the *Rapidly-Exploring Random Trees* (RRT) as the basic algorithm of this family.

Beyond basic motion planning

Particular problems go beyond the basic formulation of motion planning, and specific methods have been devised for each of them.

- **Differential constraints**. Planning with constraints on velocity and acceleration is known as *kynodinamic planning* (in particular, planning both a path and velocities along it for a robot arm is termed *trajectory planning*), and if such differential constraints cannot be integrated into derivative-free constraints -like in vehicles with limited turning radius-, we have a *nonholonomic planning* problem.
- **Multiple robots**. In a scenario composed of multiple robots, collision-free paths have to be found that allow each one of the robots reach its individual goal. Decoupled approaches, like prioritized planning or fixed-path coordination, are preferred to costly centralized solutions.
- **Moving obstacles**. It is assumed that the motion of the obstacles is known in advance. In such cases, an additional temporal dimension could be added to the configuration space, with the constraint that only forward paths along this dimension are allowed. Planning in such a space is computationally hard. Alternatively, the problem may be decoupled into a path planning and a motion timing part.
- **Manipulation planning**. Here transit and transfer modes have to be considered, the first being standard motion planning problems of the robot towards a part, the second being the robot carrying the part. The achievement of stable grasps has also to be included in the planning.
- **Assembly planning**. Planning the ways the different parts of an assembly can be brought together, respecting the precedence constraints between the parts (some part *must* be mounted before others).

- **Planning with sensing uncertainty**. This type of planning copes with limited knowledge about the configuration space. Sensor information is employed to plan in an *information space* instead, with information feedback about the current state.

Task planning

Task or action planning consists in symbolic planning in terms of statements about the world and the robot. Actions modify the current world state where they take place, and planning aims at sequencing or concatenating actions such that starting at an initial state, a goal state is achieved. This concept is transversal to scheduling, which means resource allocation (time, energy consumption, etc.) to a set of actions, so that specified deadlines are met while respecting resource limitations[9]. Planning techniques with time constraints allow to cope with the two problems simultaneously, and not in cascade as traditional approaches. However, here we will concentrate on the planning problem alone.

Deterministic action planning

First planners like STRIPS (developed by Richard Fikes and Nils Nilsson in 1971 at the Stanford Research Institute for computing simple plans for the mobile robot Shakey) were based on a propositional logic formulation of the world. States are described by sets of conditions (propositional variables), so that the initial state entails the set of conditions that are true at the beginning (all the others are assumed to be false), and the goal state the set of conditions that have to be true plus the set of conditions that must be false. The planning operators (PO) represent actions, and are represented by a quadruple that includes two sets for the preconditions (conditions that must be true and conditions that must be false in order to execute the actions), and the postconditions or effects of the action (again a set of true and another of false conditions). Planning consists then in determining a sequence of POs that change the world successively from the initial to the goal state. This planning language is deterministic in that after execution of each action, effects hold completely.

This has inspired what nowadays are known as Planning Domain Description Languages (PDDL). Algorithms based on PDDL make the planning problem tractable by resorting to simplifying assumptions, that may include[10]:

- finiteness (the domain has only finitely many objects, actions, and states)
- information completeness (the planner has all relevant information at planning time)
- determinism (actions have deterministic effects)
- instantaneousness (actions have no relevant duration)
- idleness (the environment does not change during planning)

These languages extend the propositional nature of STRIPS by upgrading to predicate logic formulations, that is, allowing the existence of non-grounded variables, besides the constants, in the describing conditions. Thus, the preconditions and effects of the POs representing actions include also free variables, and planning requires not only finding a sequence of POs but also grounding

9 Hertzberg and Chatila (n 2).
10 ibid.

consistently their variables, i.e. determining valid constant values for these variables. For this reason, a PO is now also known as *action schema*, it is a structure where different groundings are possible.

The PDDL is also a standard to which different specific languages adhere, where some additional features may exist like argument typing, equality handling, conditional action effects, and some restricted form of FOPL statements[11]. Temporal planning, that is, allowing actions to specify durations (thus overcoming the instantaneousness assumption) is a distinctive feature of the extension PDDL2.1[12].

Planning not necessarily produces a total order or linear plan, but also partial order or nonlinear plans are a possible outcome. In such plans, a set of actions and an ordering relation between them are determined, whereas unordered actions may be executed in any sequence (in some formalisms even in parallel). Classical formulations of this partial-order plan generation start with the empty plan, which contains just the initial and the goal conditions, and iteratively introduce new actions by checking if the generated conditions respect the partial ordering relations. This strategy would lead straightforwardly towards the solution if all the actions were independent, but quite frequently it appears that the effects of a newly introduced action threaten the partial ordering in the plan attained so far. A way out of such conflicting and time consuming interactions is to resort to subplans as planning macros, thus obtaining a hierarchical structure for planning, This is the idea behind hierarchical task networks (HTNs), were the plan is incomplete as long as there exist unexpanded (i.e., to the lowest level in the hierarchy) subplans.

Newer deterministic planners which have earned considerable success like GRAPHPLAN rely on the expressive power of *planning graphs* and the strength of logical inference by *planning as satisfiability*. As a drawback, the latter introduces the well-known *frame problem*, i.e., how to express changes without having to state explicitly all what remains unchanged. Alternative logic formulations like deductive logic or temporal logic have also originated quite efficient planners.

Probabilistic action planning

Modern planning approaches cope with the fact that sensors often provide incomplete information, which means that planning has to be performed under uncertainty. The standard formulation of this kind of problems is the Markovian decision processes (MDPs). To the conventional sets of states S and actions A, a new feature is added, namely that action models include conditional probability distributions for the corresponding state transitions. Typically this means to specify for each possible effect of an action a certain probability of occurrence. The aim is to obtain a *policy*, that is a function that maps states into actions, and such a policy may be derived from value iteration (VI) or policy iteration (PI) algorithms. As explained later in the context of reinforcement learning, such methods aim at maximizing the overall *utility* of the plan (where the utility of an individual action would be the negative cost associated to this action).

Like in the case of deterministic planners with PDDL, also standards have been provided in international planning competitions for probabilistic planners: PPDDL (with the first P standing for Probabilistic) and more recently RDDL (inspired in the transition models of DBNs).

One step further is to relax the complete observability assumption of the world state. This gives raise to the Partial Observability Markov Decision Processes (POMDPs), that add to the MDP formulation an *observation model*: a finite set O of possible observations that the robot can perform, as well as the conditional probabilities of making a specific observation o in state s. Refor-

11 Hertzberg and Chatila (n 2).
12 ibid.

mulating planning in the *belief space*, i.e. probability distributions over the state space corresponding to the robot's belief of being in such state after executing action a or observing o, the same search techniques as in MDPs can be applied, namely VI or PI. As belief space is exponentially larger than state space, only quite simple POMDPs can actually be tackled in this way, although also some approximations make it more tractable[13].

Finally we have to stress again that generally the robots have a fixed repertoire of actions, and if no combination of such actions produces a satisfactory fulfillment of the goal, the robot ends up concluding that no plan exists. The way out is providing the robot with the capability of expanding this repertoire. This means learning new actions.

Learning

Outside of the controlled and structured environments where most robots dwelled up to now, robots have to face a world they know nothing about. Their human programmers may provide them a declarative description of some of the world's traits. Such a formal description is obviously incomplete, inaccurate, simplistic and hardly useful for mere survival. In other words, robots have to carry out their activity in a world that is

- partially known, (i.e., incomplete knowledge about the world)
- partially observable (not even relevant observation can be taken for granted), and
- dynamic (i.e., changing).

As for the latter, one should add that changes stem either from ambient[14] phenomena or from actions performed by other agents. Some of such changes can be anticipated with reasonable assumptions on expected behaviors: ambient changes may follow physical laws or established rules, whereas agent actions may adjust to the knowledge about its goals and motivations. Such knowledge may be previously encoded in the robot's knowledge base, or it must be acquired, i.e. *learned*. Non-coded as well as unpredictable knowledge render learning as an unavoidable requisite for autonomous robots to be deployed in unstructured environments. To this end, machine learning techniques apply.

A classical but quite informative classification of learning strategies distinguishes between supervised and unsupervised methods. In **supervised** learning, there is a teacher providing feedback to the system about its learning performance, by providing the correct answer after execution of the learned action or task. This can also be expressed in terms of formulating the goal of learning in terms of computing the function f that relates a given input X with an output Y, that is, $Y = f(X)$: in supervised learning, the corresponding Y to certain X is provided by the teacher, that can supervise how in successive iterations function f is approximated increasingly well. It is also the teacher who decides when the learning system has achieved an acceptable level of performance and learning terminates. The two big families of supervised learning methods are *classification* methods (here, each output Y is a class or category, and function f has to correctly assign each individual X to its class) and *regression* algorithms (both variables are numbers and f may be an analytical function like for example in linear regression). Supervised learning can of course also take place at a symbolic level, like *inductive logic programming*, which aims at synthesizing a

13 Hertzberg and Chatila (n 2).

14 including both natural episodes as well as typical –possibly regular– events in human environments, like the alternation in traffic lights

minimal logical program that provides the correct true or false values to the corresponding input variables. Popular classification methods used in Robotics include:

- **Support Vector Machines** (SVM), which try to maximize the gap separating the data belonging to different classes (the base procedure is linear, but the *kernel trick* allows for non-linear separating functions as well).
- Statistical methods like **Bayesian learning** (an application of the Bayes rule in order to find the probability of a certain class to be the one to which the input data belong, knowing the priors of class distribution in advance and having determined the likelihoods of the data for each class in the training phase), and its variants maximum likelihood and expectation maximization.
- **Neural networks** (NN), which consist in combining basic computational units, called neurons, linked by weighted connections. Each such neuron computes the weighted sum of its inputs and fires if this sum is larger than a given threshold. In the learning phase, the weights associated to each neuron input are updated until the network performs a satisfactory classification. NN allow for online learning, but they provide no insight into the classifier.

The supervised learning paradigm *par excellence* in Robotics is Learning from Demonstration, which is described more in detail in the homonymous Section.

Unsupervised machine learning, on the contrary, provides no information about Y to the learning system. There is no teacher and the system has to determine the implicit structure underlying the input data X. That is, it tries to model such distribution or structure, and performance can be expressed with regards to how well new data adjust to the found structure. Unsupervised learning families include *clustering* methods (inputs are grouped in clusters by some proximity or partitioning criterion, k-means clustering being a popular such algorithm) and *association rule learning* (i.e., to discover rules describing large portions of input data). In Section "Reinforcement learning" we take a closer look at this widely used and typical unsupervised learning scheme in Robotics.

In the case of a robot, due to its embodiment, embedded in physical surroundings, learning heavily relies on visual perception. Perception is the input stream from which descriptions about the current state of the world can be extracted, which in turn allows to couple sensed changes in the environment to particular actions performed by the robot. Thus, before examining learning paradigms, a brief overview on perception is provided. Unless othewise stated, we will always refer to visual perception.

Perception

In the context of learning, visual perception is doubtless the most powerful input channel for a robotic system to obtain a description of the world. It can also be the most efficient one, because of the immediate encompassment of a whole scene, as long as the involved visual processes avoid becoming a computational bottleneck. Computer vision (CV for short) is about processing static images or a continuous video stream, and this includes, in broad terms, the following steps:

- Image acquisition (with digital mono or stereo cameras, maybe with enhanced features like range measuring, or signal measuring beyond the visual spectrum),
- Preprocessing (this includes several basic image enhancing processes),

- Segmentation (division of an image into regions, that may correspond to different objects or object parts),
- Recognition (bringing image regions in correspondence with object models or labels).

Applications of CV that are relevant for Robotics include object recognition (aka classification, that is, assigning a specific view of an object to its corresponding class, which is prespecified or has been learned), identification (of an individual object, face, fingerprint, iris, or the like), or detection (of an object, a defect, a person, etc. within an image). Specific instances of recognition like facial expression recognition or gesture recognition (including its dynamic version along a video sequence) are of particular interest in human-robot interaction. They are frequently used in the learning from demonstration context, as shown below. As for this application, another perception channel reveals as being quite useful, namely measuring the forces exerted on the arm (particularly in kinesthetic learning). To this end, either force/torque sensors mounted on the wrist are used, or force measurings at the robot's joints. Turning back to CV, the concurrence of different recognized objects/people or their spatial relationships may lead to quite basic instances of scene understanding, which however are still far from the richness of human scene understanding, due to the lack of knowledge about social and cultural cues.

State of the art tools are Convolutional Neural Network (CNNs), which are artificial neural networks inspired in the visual cortex of animals and which model visual perception by humans (and by animals in general). They perform very well in image recognition: they are close to humans in object classification and detection (as long as images are not altered with filters, as in current popular smartphone applications), and even slightly better in fine grained classification. The labelling of the individual images appearing in the databases used for training these CNNs have been done by humans. Stated differently, the basis of classification, the implicit criteria for categorization have been established by humans. Nonetheless, projects exist nowadays to perform such categorization automatically, from the text accompanying the images in the world wide web, like captions of the photographs appearing in the news.

Learning to act

In what follows we examine the two most extended learning techniques in Robotics, which happen to be quite genuine representatives of unsupervised and supervised learning. We will not go very deep into the technical detail, and instead try to provide a general idea of how they work, with special emphasis on the role played by the human programmer.

Reinforcement learning

Reinforcement learning (RL) refers to a set of algorithms devised to obtain an optimal or near-optimal policy (action selection based on the current state), without intervention of a teacher (i.e., they belong to the unsupervised learning category). The setting is conceived as a Markov Decision Process, the current state and action selected determine a probability distribution on future states, that is, the effect of applying an action depends only on the current state where the action is applied, regardless to the previous history. Full observability of each state is assumed, although partial observability formulations do also exist. The only feedback provided to the learner comes from the environment, where the robot's actions take place. There are many RL techniques, but the common features include the following:

- a set of environment and robot states S;

- a set of actions A that can be executed by the robot;
- policies of transitioning from states to actions;
- rules that determine the scalar immediate **reward** of a transition;
- rules that describe what the agent observes.

The reward typically comes with the observation of the last transition undergone, and expresses the degree to which the resulting state (or the action leading to this state) is desirable. The reward is provided by state observations, but it is up to the designer of the RL algorithm (or the user implementing it) to decide which environment (or robot) features are used for computing the reward. Rewards express immediate satisfaction degrees, but what really guides the learning process are the *value functions* (aka utility functions) of the transitions (or of the states), that correspond to long-term degrees of desirability. Values are computed from the rewards of the estimated optimal course of actions leading to the final goal of the learning process. A certain state (or the transition leading to it) may have a high reward but a poor value, and vice-versa. While rewards are directly taken from state observations, values must be estimated and reestimated again and again from the sequences of observations a robot makes over its entire lifetime (i.e., from the different action courses that lead to these sequences of state observations). Some RL methods use a *discount factor* associated to future rewards, which allows to tune the relative influence of immediate versus long-term desirability. It should also be noted that most RL methods are stochastic approximations of exact Dynamic Programming: instead of sweeping over the whole state space, sampling of states according to the underlying probabilistic model is performed.

Value estimation can thus be seen as central to RL. Nonetheless, evolutionary optimization methods (like genetic algorithms or simulated annealing), which search the policy space directly, could be used instead. These methods do not allow to interact directly with the environment while learning, whereas value function estimation RL does, but they can be used to contrast their results with those obtained with RL. This online use of RL raises another question, namely the *exploration-exploitation* tradeoff: exploring new, potentially more rewarding states vs. exploiting current knowledge. A typical strategy to deal with this issue (among others) is the ϵ-greedy method, where the action currently believed to be optimal is chosen with probability $1-\epsilon$, and another random action is chosen with probability ϵ.

Future projections of the system's behavior may either be model-based or model-free. The model mimics the system's behavior, it allows for simulations of possible courses of actions. An example of model-based algorithm is Adaptive Real-time Dynamic Programming. Model-free algorithms, on the other hand, do not require any knowledge about the consequences of the individual actions. Q-learning is a characteristic example of model-free RL algorithm.

Learning from demonstration

Learning from demonstration means basically that the robot is taught by performing the task to be learned "in front of" it (i.e., it is a supervised learning methodology) The *teacher* (generally a human demonstrator) executes several instances of the task in a way that the robot's perception system is able to follow their performance. If learning is conceived as taking place in a search space (the space of all the possible solutions to determining the correct trajectory or the correct sequence of actions to attain the goal), then learning from demonstration (LfD) allows to drastically reduce this space, either by focusing or restricting learning to a close neighborhood of the solution, or by pruning away the parts corresponding to wrong solutions (by counterexamples). LfD,

aka *imitation learning*, is also the way of programming robots in a natural and intuitive way. The human-robot interaction (HRI) tools used to this end will be examined below, but it is pertinent to mention now one of these tools, namely kinesthetic guidance. This refers to physically guiding a robot arm along the desired trajectory by pulling and pushing it at the end-effector (or other parts of the arm). And it is pertinent to mention it here because one of the very first industrial robot programming methods consisted precisely in guiding the robot (directly, with the help of a teach pendant -a kind of wired remote-, or by driving a lightweight mock-up) along the desired trajectory, which was registered for later reproduction in the execution phase. What distinguishes LfD from these early programming ways is that not an exact reproduction of the taught trajectory is sought, but a *generalization* over several such demonstrations, which are executed in slightly varying conditions. As a result of the learning process, such a generalization aims at adjusting to the current conditions during execution.

Skill transfer in LfD means to answer the following questions:

- What to imitate?
- How to imitate?
- Who to imitate?
- When to imitate?

The last two questions haven't received much attention in research, as in the usual setting there is just one teacher, and the instants where the demonstration begins and ends are well-established. *What to imitate* refers to determine which are the relevant parts of the demonstration that need to be learned. This is achieved by the repeated demonstrations in the learning phase: only the relevant parts of the task are expected to be maintained along the demonstrations (thus, certain variability is desirable). Furthermore, and this is an HRI issue, *social cues* may be used for focusing the attention on the important parts of the task: gazing or pointing at region and time-intervals of interest, using verbal statements, etc. This means of course that the robot has to be previously endowed with the capacity of interpreting such social cues. Furthermore, it has to be established whether the robot is intended to reproduce the articular motions of the teacher, to follow the trajectory of the hand, or if only the final position attained is relevant. The first variant, pursued e.g. in gesture learning, may be impossible due to large differences in the embodiment of teacher and robot (see the correspondence problem below). In any case, also a metric of imitation performance has to be defined, associated to the different alternatives the imitation may be conceived, namely as achieving the same final relative position, the same absolute position, or the same relative displacement as in the demonstration.

How to imitate addresses the so-called *correspondence problem*, which relates to the different embodiment of teacher and learner, therefore exhibiting a different kinematic structure (such differences may be just a question of scaling, or of different proportions, or even of different number, disposition or type of joints).

In the research community, LfD is usually distinguished as occurring at trajectory level or at task (or symbolic) level, and in the previous paragraphs we have been switching indistinctively between both modalities. At trajectory level, the robot learns to perform basic sensory-motor skills, and for this reason it can also be envisioned as learning control policies. It is a generalization of movements, aiming at obtaining a generic representation of motion which allows to encode very

different types of signals/gestures[15]. The what-to-imitate problem translates into which variables have to be chosen to encode a movement. Alternatives addressed by researchers include encoding at joint level, at task level, or in torque space. The type of motion to be encoded may be cyclic (i.e., repeated under slightly varying conditions), discrete, or combining both types. Prior to the actual learning, dimensionality reduction techniques may help to reduce its computational load. The original recorded signals are projected onto a latent space with fewer dimensions while preserving the maximum amount of information. A typical such technique is Principal Component Analysis (PCA), where the main direction to be projected upon is the one where the data exhibit the highest variance, the next one is the following highest variance direction, with the constraint of being orthogonal to the first one, and so forth. The encoding may be based on statistical analysis (like Gaussian Mixture Models, combined with Gaussian Mixture Regression for the predictions (GMM/GMR), Hidden Markov Models (HMMs), or other), or based on dynamic systems, like Dynamic Motion Primitives (DMPs), Locally weighted regression (LWR) or even recurrent Neural Networks[16]. Vision and proprioceptive sensors are the most common input sources, although force/torque sensors have also been considered in recent years[17], especially for the cues they provide in collaborative tasks[18]. As for the latter, robots involved in collaborative tasks, the assignment and switching of the roles of master (teacher) and follower within the same task have also been studied based on interaction forces[19], or switching between reactive and proactive behaviors by anticipating human motions[20].

As for task level learning, the aim is to formulate the task in terms of predefined atomic actions (or small standardized sequences of atomic actions) represented symbolically, for example as rules in STRIPS-like formalisms. Such rules, if appropriately learned, can be concatenated afterwards by using a planner to reproduce the sequence of actions that fulfills the task under slightly different start and goal conditions. Such sequences of actions can also be encoded and reproduced using classical machine learning algorithms like HMMs, or graph-based hierarchical encodings[21]. Symbolic learning requires the sensory input to be processed and segmented into meaningful world transitions corresponding to the aforementioned actions. Sequencing entails learning some kind of precedence constraints between actions. Some actions have to strictly precede others, and this is discovered after a sufficient number of demonstrations whose variability uncovers such strict precedence, distinguishing them from actions that only circumstantially happen to occur in a given temporal order. While task or symbolic learning allows to learn interactively high-level skills, these methods have the disadvantage of relying on a large amount of prior knowledge (e.g. about the basic atomic actions) in order that the demonstrated sequences can be segmented consistently.

In LfD, the more demonstrations provided to the robot, the better should the task at hand be

15 Aude Billard and others, 'Robot Programming by Demonstration' in *Springer Handbook of Robotics* (2008) ⟨http://dx.doi.org/10.1007/978-3-540-30301-5_60⟩ .

16 Masato Ito and others, 'Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model' (2006) 19(3) Neural Networks 323 ⟨http://dx.doi.org/10.1016/j.neunet.2006.02.007⟩ .

17 Leonel Rozo, Pablo Jiménez, and Carme Torras, 'A Robot Learning from Demonstration Framework to Perform Force-based Manipulation Tasks' (2013) 6(1) Intell. Serv. Robot. 33 ⟨http://dx.doi.org/10.1007/s11370-012-0128-9⟩ .

18 L Rozo and others, 'Learning Physical Collaborative Robot Behaviors From Human Demonstrations' (2016) 32(3) IEEE Transactions on Robotics 513 ⟨http://dx.doi.org/10.1109/TRO.2016.2540623⟩ .

19 Y Li and others, 'Role adaptation of human and robot in collaborative tasks' (May 2015) ⟨http://dx.doi.org/10.1109/ICRA.2015.7139983⟩ .

20 W Sheng, A Thobbi, and Y Gu, 'An Integrated Framework for Human-Robot Collaborative Manipulation' (2015) 45(10) IEEE Transactions on Cybernetics 2030 ⟨http://dx.doi.org/10.1109/TCYB.2014.2363664⟩ .

21 Billard and others (n 15).

learned by the robotic system. However, the more demonstrations have to be shown to the robot, the more annoying and tedious becomes teaching the task to the robotic unit. Ideally a few demonstrations should suffice, as long as they are different enough as to highlight the really significant traits of the task, as said above. But, with the exception of very simple cases, it is difficult in general to design such a set of significant demonstrations. An interesting way out is to resort to some kind of *incremental* learning. Rough versions of the task are learned with as few demonstrations as possible, and the robot starts executing them right away. Performance may be poor at the beginning, but monitoring of the robot by the programmer or the user allows to detect where improvements are required. The learned task is progressively refined by providing new demonstrations. The errors observed in early performances allow the teacher to identify where the new demonstrations have to provide new insights about the task. Verbal and non-verbal cues can be used by the teacher to guide the attention of the robot system towards the parts of the task that need to be improved. This guided incremental learning is often called *scaffolding* or *moulding* of the robots knowledge about the task, it can alse be seen as a variant of *coaching*.

Driving the attention towards some parts of the task or specific locations of the setting is a typical application of HRI. Social cues have been investigated and applied to this end. They encode in some sense priors of the statistical learning methods, speeding up learning. Non-verbal cues include pointing and gazing, whereas verbal instructions require some form of natural language processing. Even the prosody of spoken instructions has been studied in the search of such social cues. As said above, CV techniques are needed for interpreting the gestures associated to such attention-driving cues.

Last but not least, another way of avoiding a huge number of demonstrations is transferring the refinement of action learning to an unsupervised method. This has the advantage of reducing drastically the search space of the latter, thanks to the demonstrated tasks, while avoiding to further resort on the teacher for obtaining a more accurate performance. Typically, such unsupervised task refinement learning consists in some form of RL. In[22] the perspective is somehow complementary: a (relational) RL approach is enhanced with occasional requests to the teacher, who performs demonstrations oriented at pruning the search space significantly. The own system provides suggestions on which aspects should be covered by the demonstration. The idea behind this approach is that the time of the human teacher is much more valuable than the robot's time and thus requests should be kept at a minimum.

Finally it should be stressed that some LfD schemes resort to biological analogues, being the most characteristic those models that try to mimic the functioning of Mirror Neuron Systems, which are responsible of imitation in animals.

Conclusions

In this contribution we have presented a brief overview on the most salient cognitive techniques that can provide robots with a certain degree of autonomy, and some kind of smart response in front of a continuously changing world, with predictable evolutions but also surprising contingencies. In first place we have perception, the input to the control system and thus to decision making. Perception depends on the identification parameters provided by the human designers/programmers, and modern classification algorithms like the powerful CNNs operate on labeled data (or contextual information). In other words, the semantic content has been previously given by humans. This

22 David Martínez, Guillem Alenyà, and Carme Torras, 'Relational reinforcement learning with guided demonstrations' [2016] Artificial Intelligence ⟨http://dx.doi.org/10.1016/j.artint.2015.02.006⟩ .

statement can also be extended to methods learning from data extracted from internet, with the potential risks attached to such an uncontrollable source.

In simple reactive controls the response is always perfectly defined, and can be reliably predicted, up to errors or failures. The problem becomes more involved when higher cognitive functionality is invoked. Motion planners compute paths of the robots (maybe trajectories, if the different velocities along the path are also considered) and even such an apparently innocuous duty can have social or ethical implications if, for example, such a path appears traversing a sensible area. The geometrical basis of such planners excludes any moral responsibility of the robotic agent, as it corresponds to the human programmer to exclude such areas from free-space, or to foresee their possible existence if the system builds its maps autonomously, and in this case, the means for identifying such areas should be provided.

As for task planning, we have seen that deterministic planners rely on logical linking of actions. Of course this can become arbitrarily complex, but at least **traceability** of plans is granted. Undesired side-effects from some actions can be traced back for uncovering the conditions that produce them, which in turn may have consequences on the design of the actions (more than on the process of planning per se). Probabilistic task planners, on the other hand, are more difficult to trace back-if not impossible- due to the randomness of some decisions[23]: the probability of occurrence of each effect is naturally known, but the actual effect that finally takes place is clearly unknown until it happens. Such probabilities may provide a quantization of the liability of the human designer to each outcome, a kind of calculated risk, but the chaining of several actions may introduce rapidly a high degree of complexity, as for the combinatorics of effects. Moreover, it is a common practice to include a number of spurious effects within a generic "noise" effect with certain probability.

We have seen the two paradigmatic learning strategies. Supervised learning, and in particular, learning from demonstration clearly assigns the whole responsibility of what is learned to the teacher, the one who provides the demonstrations. Liability may be limited by the learning performance of the system, and the errors that may arise during the learning process. In unsupervised learning the issue is a little bit more tricky, although what is considered a reward to guide the learning development is of course a decision of the designer. Even the fact that it is long-term value function and not the immediate reward, as we have seen, what determines action selection, the computation of the value function still bases on what the designer has considered to be the state variables to promote. Model-based learning even allows for a certain kind of predictions.

In sum, cognitive processes in robotic systems have been designed and implemented by humans. Deterministic processes can be predicted, as for their evolution and final result, or traced back. In the case of probabilistic processes, also certain predictions on their behavior can be made, with associated probabilities of occurrence, although combinatorics and insufficient computing power may pose some limits to the designer's or programmer's anticipation capabilities. This can be partially alleviated by performing a worst-case analysis, which reduces the options to consider. In any case, it is up to the human programmers/users to decide whether complexity and uncertainty may shelter questionable decisions of the robotic system. As for today, there is nothing like an autonomous will of a robotic system, and the morality of its actions is the morality of its human designers, programmers or users.

23 unless, of course, a register of all the history, i.e., of the whole sequence of states and actions, is kept, and not just the final result

Acknowledgments

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness under project RobInstruct (TIN2014-58178-R).

References

Baader F, Horrocks I, and Sattler U, 'Description Logics' in Staab S and Studer R (eds), *Handbook on Ontologies* (Springer Berlin Heidelberg 2004) ⟨http://dx.doi.org/10.1007/978-3-540-24750-0_1⟩.

Billard A and others, 'Robot Programming by Demonstration' in *Springer Handbook of Robotics* (2008) ⟨http://dx.doi.org/10.1007/978-3-540-30301-5_60⟩.

Hertzberg J and Chatila R, 'AI Reasoning Methods for Robotics' in *Springer Handbook of Robotics* (2008) ⟨http://dx.doi.org/10.1007/978-3-540-30301-5_10⟩.

Ito M and others, 'Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model' (2006) 19(3) Neural Networks 323 ⟨http://dx.doi.org/10.1016/j.neunet.2006.02.007⟩.

Li Y and others, 'Role adaptation of human and robot in collaborative tasks' (May 2015) ⟨http://dx.doi.org/10.1109/ICRA.2015.7139983⟩.

Martínez D, Alenyà G, and Torras C, 'Relational reinforcement learning with guided demonstrations' [2016] Artificial Intelligence ⟨http://dx.doi.org/10.1016/j.artint.2015.02.006⟩.

Matarić M, 'Learning in behavior-based multi-robot systems: Policies, models, and other agents' (2001) 2(1) Cognitive Systems Research 81 ⟨http://dx.doi.org/10.1016/S1389-0417(01)00017-1⟩.

Rozo L, Jiménez P, and Torras C, 'A Robot Learning from Demonstration Framework to Perform Force-based Manipulation Tasks' (2013) 6(1) Intell. Serv. Robot. 33 ⟨http://dx.doi.org/10.1007/s11370-012-0128-9⟩.

Rozo L and others, 'Learning Physical Collaborative Robot Behaviors From Human Demonstrations' (2016) 32(3) IEEE Transactions on Robotics 513 ⟨http://dx.doi.org/10.1109/TRO.2016.2540623⟩.

Sheng W, Thobbi A, and Gu Y, 'An Integrated Framework for Human-Robot Collaborative Manipulation' (2015) 45(10) IEEE Transactions on Cybernetics 2030 ⟨http://dx.doi.org/10.1109/TCYB.2014.2363664⟩.

Tenorth M and Beetz M, 'KNOWROB knowledge processing for autonomous personal robots' (October 2009) ⟨http://dx.doi.org/10.1109/IROS.2009.5354602⟩.

Tenorth M and others, 'Representation and Exchange of Knowledge About Actions, Objects, and Environments in the RoboEarth Framework' (2013) 10(3) IEEE Transactions on Automation Science and Engineering 643 ⟨http://dx.doi.org/10.1109/TASE.2013.2244883⟩.