# High-level robotic control: beyond planning
# Position paper

**Hector Levesque and Ray Reiter**
Dept. of Computer Science
University of Toronto
Toronto, Canada. M5S 3H5
http://www.cs.toronto.edu/~cogrobo/

We agree with the premise of this workshop that it is time to consider robotic systems that integrate the results of various independent research efforts. Our position is that to achieve this integration in a principled way, it will be necessary to take a higher level view of robotics, one that takes quite seriously the idea that a robot must not only sense and act in the world, but reason about itself and its environment. In this position paper, we motivate and describe in an informal way the direction we are taking in this regard. Technical details can be found in the papers at our website above.

For the past five years or so, a group of us[1] at the University of Toronto have been engaged in what we call *Cognitive Robotics*, which we take to be the study of the knowledge representation and reasoning problems faced by an autonomous robot (or agent) in a dynamic and incompletely known world. Central to this effort is to develop an understanding of the relationship between the knowledge, the perception, and the action of such an robot. The sorts of questions we want to be able to answer are

- to execute a program, what information does a robot need to have at the outset vs. the information that it can acquire *en route* by perceptual means?
- what does the robot need to know about its environment vs. what need only be known by the designer?
- when should a robot use perception to find out if something is true as opposed to reasoning about what it knows was true in the past?
- when should the inner workings of an action be available to the robot for reasoning and when should the action be considered primitive or atomic?

and so on. With respect to robotics, our goal (like that of many in AI) is *high-level robotic control*: develop a system that is capable of generating actions in the world that are appropriate as a function of some current set of beliefs and desires. What we do *not* want to do is to simply engineer robot controllers that solve a class of problems or that work in a class of application domains. For example, if it turns out that online reasoning is unnecessary for some task, we would want to know what it is about the task that makes it so.

In our opinion, previous attempts at high-level control within AI have been hampered by an over-reliance on automated *planning*: given a goal to achieve together with a description of some initial state of the world, and the prerequisites and effects of a set of primitive actions, find a sequence of actions that satisfy the goal, and then hand them over to the robot for execution. This suffers from some serious drawbacks:

- *no sensing*: the planning system is expected to generate a sequence of actions without considering the results of sensing;
- *lack of reactivity*: exceptional situations might arise during during execution: high-priority interrupts, failures of the execution modules, unanticipated situations;
- *computational intractability*: for all but very simple domains, automated planning appears to be infeasible; at its very best, planning seems ill-suited to generating very long sequences of actions;
- *incompatibility with conventional robotics*: conventional robotics deals with micro-actions where decisions are made many times per second in worlds characterized by noise and uncertainty.

One of the reactions to these difficulties has been the rise of so-called "insect robotics", where high-level reasoning is effectively abandoned completely in favour of sensor-based reactivity.

Our approach preserves the high-level control aspect, but reduces the dependency on automated planning. Instead of taking as input a goal that needs to be achieved and calling on a planner to generate primitive actions to achieve it, we imagine a system that takes as input a *high-level program* that needs to be executed, and calls on a *program interpreter* to generate primitive actions to execute it.

[1] Todd Kelly, Yves Lespérance, Fangzhen Lin*, Jeff Lloyd, Sheila McIlraith*, Daniel Marcu, Javier Pinto*, Shane Ruman*, Richard Scherl*, Steven Shapiro, Mikhail Soutchanski, Eugenia Ternovskaia ($*$ = *emeritus*).

By a high-level program, here, we mean a program that tells the robot or agent what needs to be done, with the following characteristics:

- the most primitive statements in the program are the external primitive actions available to the robot:
  - *move to the desk* and then *pick up the package*;
- the tests within the program pertain to conditions in the world that are affected by the robot (and other robots):
  - if *the door is locked* then ... else ...;
  - while *there is a package on the table* do ...;
  - ... after which *you must be located in Smith's office*;
- programs may be *nondeterministic*: they may contain choice points where the interpreter must make a reasoned (non-random) selection that correctly satisfies some later constraints
  - *go through the appropriate door* and retrieve the package that is waiting;
  - *either go left or right as appropriate* and then ... at which point you must be located in the hall.

So high-level programs resemble plans but are considerably more general. For one thing, they can contain loops, recursive procedures, and more recently for us, concurrent actions and prioritized interrupts. Most importantly, because of the nondeterminism, they cannot be executed "blindly." It is the task of the program interpreter to figure out how to execute them. To do so, the interpreter needs to be able to determine what tests are true or false, and what primitive actions are possible or impossible, after the execution of various primitive actions. Moreover, to handle any nondeterminism, it needs to search through various possibilities to find a sequence of primitive actions that satisfies all the constraints embedded in the program.

In searching for a sequence of actions, what an interpreter does is not so different from planning, but for a very specific sort of goal: get to a final state where the given high-level program has been successfully executed. There is, however, a crucial difference between this search and the search required in traditional planning: a high-level program typically provides strong clues about what the desired sequence of primitive actions should be like. In fact, when the high-level program is (almost) deterministic, the program interpreter requires (almost) no search.

Of course one can write high-level programs that are so nondeterministic, that nothing is gained over planning. Consider for example,

> until the goal $G$ is achieved, do an appropriate primitive action.

In this case, the program interpreter must consider all possible sequences of actions, just as a planner would, and would have the same computational problems. One would never expect to be able to generate a sequence involving (say) 1000 or 10,000 steps this way. (In the blocks world, even a 100-step plan seems infeasible.) But this is a pathological case. Typically, the user considers what needs to be done and writes high-level programs accordingly. The downside is that a user has to focus on the procedural aspects of the goals to be achieved; the upside is that we can provide high-level control in applications whose complexity goes well beyond the range of automated planning.

Although our interests lie primarily in the theoretical foundations of these ideas, we have designed a high-level programming language embodying them (called Golog) and implemented a program interpreter for it. Already a number of applications have been built in Golog and its successor ConGolog:

- a simulated elevator controller;
- a demonstration mail delivery robot at Toronto and York Universities (see below);
- a robot museum guide at the University of Bonn (Germany) that is controllable online via the Web; (The robot in this case is an RWI-B21 robot running Golog and lower level Rhino software.)
- business process simulation; (Here Golog is used as a tool to analyze business processes, rather than to execute them.)
- characters for computer animation; (Golog is used to specify the behaviour of characters, from which realistic graphical animations are then generated.)
- a softbot application in the personal banking domain. (In this case, Golog is used to implement agents which run under Unix and communicate over TCP/IP. This application is significant because of its size: over 40 pages of Golog code, clearly outside the reach of planning systems.)

The mail delivery system mentioned above was our first experience with an actual (non-simulated) robot, an RWI-B21. From a high-level perspective, the main issue to be resolved was what we took to be the primitive actions. The assumption made in Golog (and in planning, for that matter) is that when the prerequisites of a primitive action are true, the action can be reliably executed by some low-level module. So for instance an action like *go to position p* may not be desirable since so many unanticipated things can go wrong. A better primitive might be something like *start going to position p*, which initiates the activity. Sensing actions can then assess the progress of the motion within Golog and appropriate actions can be taken to correct any problems.[2] Indeed, one of the major strengths of this high-level approach to robotics

[2] In ConGolog, an interrupt can be triggered when some termination condition (successful or unsuccessful) is made true exogenously.

is the ability to recover from failures (such as doors being closed unexpectedly) in a way that takes into account the current goal and what is known about the current situation.

The resulting mail delivery system was actually run on two separate robotic systems: an RWI-B21 at the University of Toronto and a Nomad 200 at York University. This is significant as it is the first time (to our knowledge) that a robotic control program is run successfully on two separate platforms from different manufacturers and with quite different low-level software environments.

Our sense from our admittedly limited experience with these systems is that we may be seeing a shift in robotics research. Up until quite recently, "robot programming" as such was tackled mainly by engineers and hobbyists, and required considerable familiarity with the workings (and especially the failings) of the underlying hardware.[3] Increasingly, as a result of much more stable robotic platforms and much better low-level interfaces and simulations, it has become possible to develop robot programs like the mail delivery system that are more portable and hardware independent. We feel that this trend will continue, and that robotics research will develop much the way computer science has, and end up dealing more and more with the sort of high-level issues we have described here.

[3] This is not unlike the situation with the first computers and computer programming in the forties.