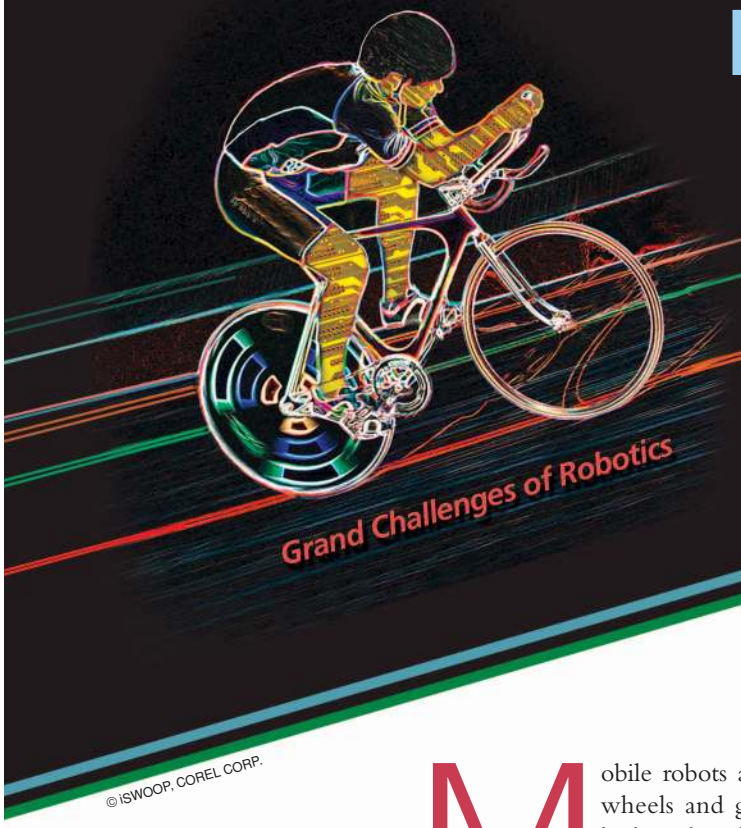
Symbolic Planning and Control of Robot Motion

Finding the Missing Pieces of Current Methods and Ideas



BY CALIN BELTA, ANTONIO BICCHI, MAGNUS EGERSTEDT, EMILIO FRAZZOLI, ERIC KLAVINS, AND GEORGE J. PAPPAS

Mobile robots are complex systems that combine mechanical elements such as wheels and gears, electromechanical devices such as motors, clutches and brakes, digital circuits such as processors and smart sensors, and software programs such as embedded controllers. They have mechanical constraints (e.g., a car-like robot cannot move sideways), limited energy resources, and computation, sensing, and communication capabilities. They operate in environments cluttered with possibly moving and shape changing obstacles, and their objectives can change over time, such as in the case of appearing and disappearing targets. Robot motion planning and control is the problem of automatic construction of robot control strategies from task specifications given in high-level, human-like language. The challenge in this area is the development of computationally efficient frameworks allowing for systematic, provably correct, control design accommodating both the robot constraints and the complexity of the environment, while at the same time allowing for expressive task specifications.

A typical motion task for one robot moving in an environment with obstacles is simply stated as "go from A to B and avoid obstacles." A common approach to this problem is based on a hierarchical, three-level process. At the first level, the obstacle-free configuration space of the robot is partitioned into cells, and adjacency relations between cells are determined. The result is presented in the form of a graph. Since any path connecting the cell containing A to the cell containing B in this graph satisfies the specification (i.e., it avoids obstacles), we call this the *specification level*. In the second step, a path on this graph is chosen, for example using an optimality criterion penalizing the travelled distance and/or the proximity to obstacles. We will call this the *execution level*. Finally, in the third step called the *implementation level*, a reference trajectory traversing the sequence of cells given by the path is generated, and robot controllers are constructed so that the reference trajectory is followed.

The so-called symbolic approaches that we discuss in this article fit into the three-level hierarchy presented above, and draw upon well-established concepts in related areas, such as

1070-9932/07/$25.00©2007 IEEE

behavior-based robotics and hybrid control systems. As we enrich the language (at the specification level) and formally take into account real-world robot control, sensing, and communication constraints, concepts and tools from the theory of computation such as automata and languages arise naturally, hence the name "symbolic."

For example, the previous motion specification "go from A to B and avoid obstacles" is not rich enough to describe a large class of tasks of interest in practical applications. The accomplishment of the mission might require the attainment of either A or B, convergence to a region ("reach A eventually and stay there for all future times"), visiting targets sequentially ("reach A, and then B, and then C"), or surveillance ("reach A and then B infinitely often"). Such specifications consisting of logical and temporal statements translate naturally to formulas of temporal logics (see the following for an informal introduction to such a logic), which are traditionally used to specify correctness and safety properties of digital circuits and software programs. A language (set of words) satisfying such a formula is in general accepted by an automaton, which can seen as a generalization of a graph. If this automaton is then synchronized with a labeled graph capturing the transitions that the robot system can take between adjacent cells in the environment, then the set of all solutions at the specification level can be represented as an automaton, as opposed to a graph as described previously (see the top level of the hierarchy in Figure 1).

The search performed at the execution level becomes more involved than path finding, and is related to the classical problem of model checking in formal analysis. For example, if the

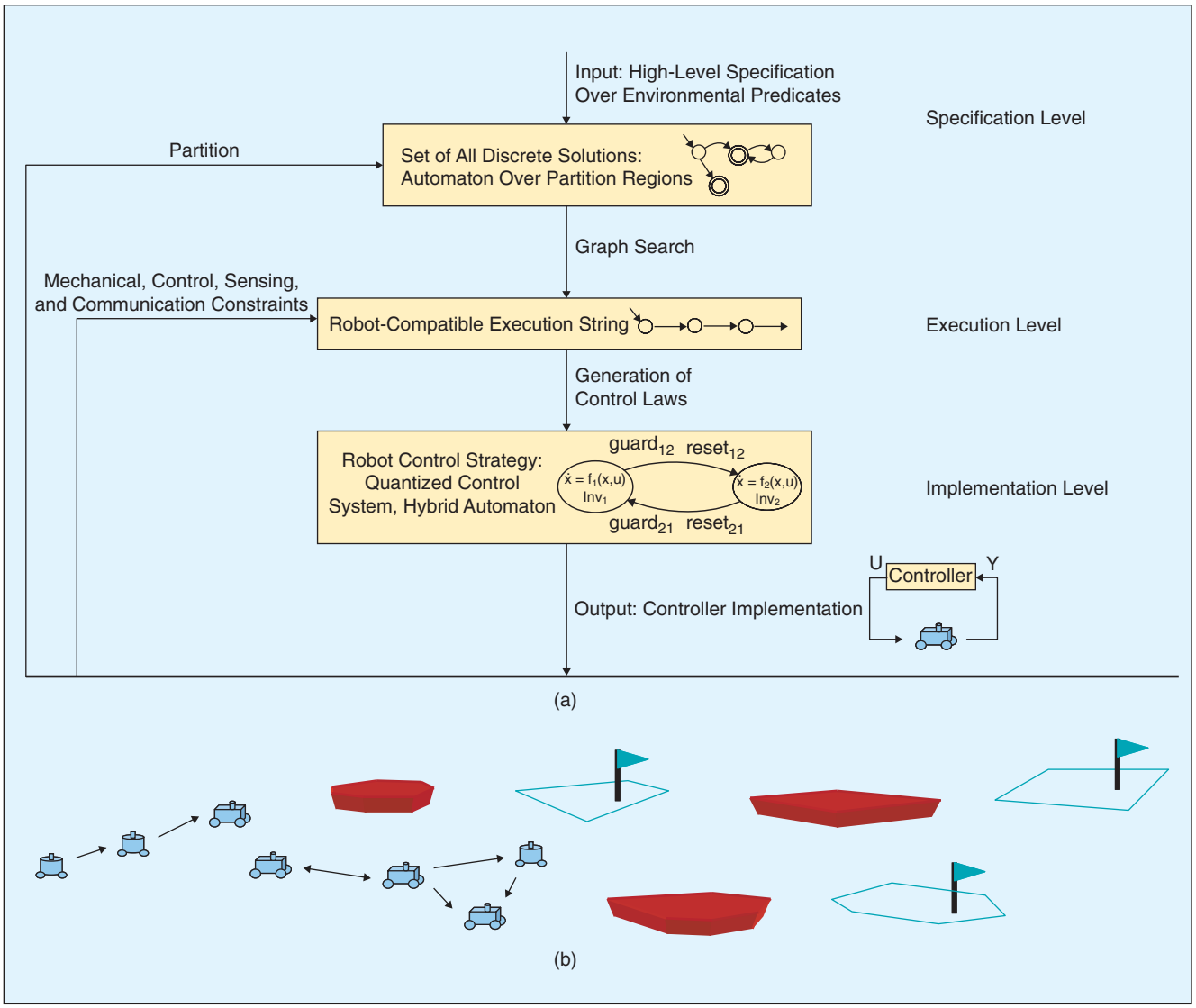


Figure 1. *(a) Hierarchical abstraction and computation architecture. A high level specification, such as a temporal logic formula over environmental predicates, together with a discrete graph representation of the environment, produces the set of all possible discrete solutions to the problem. A discrete execution is selected by taking into account the robot constraints, and then implemented as a hybrid automaton giving the control strategy for each robot. (b) A heterogeneous team of robots moving in an environment with obstacles and targets. The directed graph connecting the robots shows the communication architecture of the team.*

specification is given in linear temporal logic (see the following), this process involves a search for strongly connected components in a graph. As before, the result is an execution string (e.g., a sequence of regions to be visited by the robot), which can now be infinite (second level of the hierarchy in Figure 1). The alphabet of such a string is a finite set of control symbols or strategies, which can be determined by the partition of the environment or by predefined behaviors. The alphabet and the syntactical constructs over it are further restricted by control and sensor quantization, mechanical and communication constraints, and complexity limits.

A typical scenario arising in robot motion planning and control is shown for illustration at the bottom of Figure 1. Roughly, a heterogeneous team of robots is required to visit targets (regions marked with flags), while avoiding obstacles (red regions). At the same time, the robots might be required to assemble into one or more formations of desired geometry, such as line, triangle, hexagon, etc. The specification language should be rich and natural (e.g., allowing for statements as the one given previously). The obstacles can move and the targets can appear and disappear randomly in time and space. The robots have mechanical, control, and sensing constraints. The team is also subject to (possibly time-varying) communication constraints, illustrated as the graph connecting the robots in Figure 1. We ask the following question: "can we develop a computational framework allowing for specifying such a task in a high-level, human-like language, with automatic generation of provably correct robot control laws?"

Throughout the rest of this article, we discuss existing results providing partial answers to the above question, and emphasize the remaining open issues and challenges. The discussion is focused on the notion of discretization, which enters the problem at all levels of the hierarchy shown in Figure 1. An environment-driven discretization is induced when a robotic mission is specified geometrically in terms of regions of interest in the environment. Such a discretization, when supported by the existence of low-level controllers driving the robots through the resulting regions, can be successfully used to provide a solution to our problem for robots with simple dynamics moving in static, a priori known environments. For robots with complicated dynamics moving in dynamically changing, possibly unknown environments, the discretization is more appropriate at the level of controllers rather than environment, leading to control-driven discretizations. For example, a number of behaviors (or closed-loop control laws) can be concatenated in order to negotiate an unknown environment or cope with robot mechanical constraints.

Environment-Driven Discretization

Throughout this article, we will assume that the dynamics of a robot are described by a control system of the form

$$\dot{x} = f(x, u), \qquad x \in X, u \in U, \tag{1}$$

where x is the state of the robot, u is its control input, X is the state space, and U is the control constraint set.

Let us first focus on the implementation level of the hierarchy proposed at the top of Figure 1, and ask the following question: Given an execution string (i.e., a sequence of adjacent triangles to be visited by a robot in finite time) over regions in a partitioned environment, can we generate a robot control strategy implementing it? In other words, can we generate a control strategy driving the robot through the given sequence in finite time, while keeping it in the last region for all future times?

Consider for example the sequences of triangles $\Delta_1, \Delta_2, \ldots, \Delta_9$ from Figure 2(b), which can be seen as an execution string over a triangulated environment such the one shown in Figure 2(a). A possible solution is shown as the sequence of vector fields from Figure 2(b), where feedback controllers $u_{\Delta_i}(x)$ are assigned to each triangle Δ_i. More formally, the resulting closed-loop system is a hybrid automaton (as illustrated at the implementation level in Figure 1) $\dot{x} = f(x, u_{\Delta_i}(x))$, $x \in \Delta_i$, whose trajectories traverse the sequence $\Delta_1, \Delta_2, \ldots, \Delta_9$ in finite time, with convergence inside Δ_9. This strategy has been shown to work for any sequence of adjacent triangles and for a robot with affine dynamics (i.e., $f(x, u) = Ax + b + Bu$ in (1), which includes the fully actuated case $f(x, u) = u$) in [1]. Also, the vector fields in adjacent triangles can be matched on the separating facets, so that the produced vector field is continuous everywhere, which results in smooth robot trajectories. The computation of controllers consists of polyhedral operations only, which can be efficiently performed.

Let us now see if richer specifications can also be handled in a similar automatic fashion. Assume, as before, that the dynamics of a robot are described by an affine system. However, we consider that the specifications are given as temporal and logic statements over polyhedral regions in the environment. Consider, for example, that the robot moves in an environment with three obstacles o_1, o_2, o_3 and three targets r_1, r_2, r_3 that need to be surveyed (visited infinitely many times). In other words, we consider the following task: "*Always avoid obstacles* o_1, o_2, o_3 *and visit regions* r_1, r_2, r_3, *in this order, infinitely often.*" This specification immediately translates to a formula of linear temporal logic (LTL) over the set of symbols o_1, o_2, o_3, r_1, r_2, r_3. Informally, LTL formulas are made of temporal operators, Boolean operators, and atomic propositions connected in any sensible way. Examples of temporal operators include **X** (next time), **F** (eventually, or in the future), **G** (always or globally), and **U** (until). The Boolean operators are the usual $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\Rightarrow$ (implication), and $\Leftrightarrow$ (equivalence). The atomic propositions are properties of interest about a system, such as the set of obstacles and targets in our example. Using this notation, the task can be formally written as the following LTL formula:

$$\mathbf{G}(\mathbf{F}(r_1 \wedge \mathbf{F}(r_2 \wedge \mathbf{F} r_3)) \wedge \neg(o_1 \vee o_2 \vee o_3)). \tag{2}$$

The semantics of LTL formulas are given over labelled transition graphs (also called Kripke structures, or transition systems). In our example, such a transition system is obtained

The challenge in this area is the development of computationally efficient frameworks allowing for systematic, provably correct, control design accommodating both the robot constraints and the complexity of the environment, while at the same time allowing for expressive task specifications.

from the dual graph of the partition induced by the regions of interest, if the vertices are labelled according to their being part of obstacles or of targets, and if the edges are seen as transitions that a robot can take. We say that a robot can take a transition from a region q_i to a region q_j if we can design a feedback controller $u_{q_i q_j}(x)$ driving the robot in finite time from region q_i to region q_j through the separating facet, irrespective of the initial position of the robot in q_i. State q_i has a self transition if we can design a feedback controller $u_{q_i q_i}(x)$ keeping the robot in region q_i for all times, irrespective of the initial position of the robot in q_i. As before, the computation of such controllers can be performed by polyhedral operations.

To find initial states and control strategies from which the specification can be accomplished, one can now use a method closely related to LTL model checking [2]. Specifically, this procedure produces an automaton exhaustively describing all possible solutions to the problem. This corresponds to the "set of all discrete solutions" shown in the top box at the "specification level" in Figure 1. In this automaton, a run can be selected, which corresponds to the "execution level" in Figure 1. For such a run, an implementation is found in the form of a hybrid automaton, which associates one (or several) control laws of the form $u_{q_i q_j}(x)$ for each region q_i. This corresponds to the "implementation level" in Figure 1.

It is important to note that all the steps in the framework presented previously are performed automatically: Given the specification in terms of a formula over linear predicates in the environmental coordinates, the robots control strategy is automatically determined. The approach also has the advantage that it is robust, in the sense that small perturbations in measurements of robot positions do not affect the overall produced motion. On the negative side, the method is not complete in the sense that, if a solution is not found, it might still exist.

Even though these results suggest that automatic control of robots from high level specifications given as formulas of some temporal logic is useful and possible, several fundamental questions remain to be answered. For example, it is not at all clear that LTL is the right specification language. There are specifications (such as "proposition π holds infinitely often on all runs") which cannot be expressed in LTL, but can be accommodated by the incomparable logic computation tree logic (CTL). It is also possible that LTL is too expressive, and an unnecessarily large amount of time is spent for model checking-type analysis. Simpler fragments of LTL might be enough, as suggested in [3]. The main challenge when

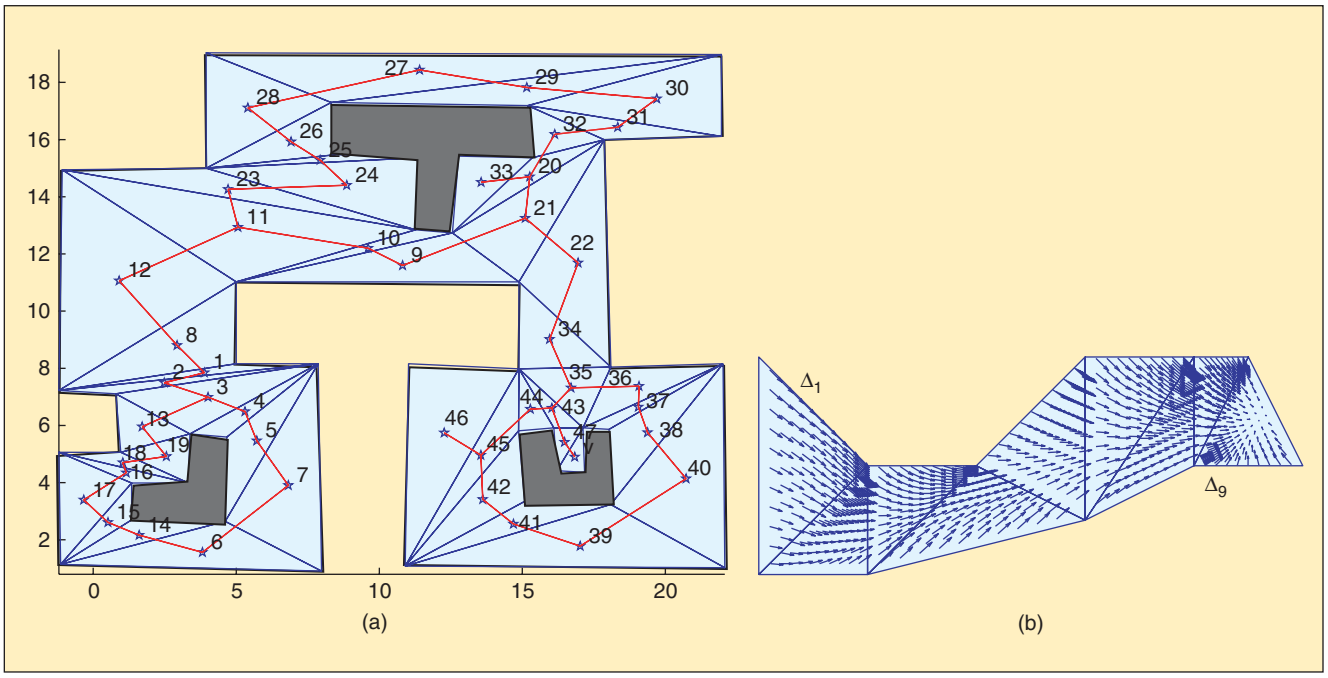


Figure 2. *(a) Triangulation of the free space in a polygonal environment and the dual graph. (b) String of nonrepetitive triangles in a triangulated environment is executed by constructing affine vector fields in each triangle.*

choosing a specification language is to find a good compromise between the expressivity of the language and the complexity of the analysis and synthesis algorithms.

Extending the aforementioned methods to more realistic robot models will raise some nontrivial questions that should be addressed in the near future. First, controllers guaranteeing robot transition from one region to another or making a region an invariant for a robot have not yet been developed for robots with nonholonomic (nonintegrable differential) constraints (see the following). Second, this approach should take into account constraints induced by digital controllers and sensors such as finite input and output spaces.

Finally, and arguably the most challenging open question is the following: "given a team of locally interacting robots, and a high level (global) specification over some environment, how can we automatically generate provably correct (local) control strategies?" What global (expressive) specifications can be efficiently distributed? How should we model local interactions (e.g., message passing versus syncronization on common events)?

Control-Driven Discretizations

The approach presented in the previous section uses environment discretization to capture the complexity of the environment. While allowing for a rich specification language over the partition regions, it is (in current form) restricted to static, a priori known environments and simple robot dynamics, such as fully actuated or affine dynamics with polyhedral speed constraints. Robots with more complex dynamics such as unmanned ground car-like vehicles and air helicopter-like vehicles might not be able to implement executions strings over partition regions. In this situation, the discretization may be more appropriate at the level of controllers rather than environments. The argument behind such a control-driven discretization is that the global control task can be broken down into more easily defined behavioral building-blocks, each defined with respect to a particular subtask, sensing modality, or operating point. Strings over such behaviors make up words in so-called motion description languages (MDLs) [4]. An example of such a string is $(k_{i_1}, \xi_{i_1}), \ldots, (k_{i_q}, \xi_{i_q})$, where $k_{i_j} : \Re^+ \times X \to U$ are feedback control laws and $\xi_{i_j} : \Re^+ \times X \to \{0, 1\}$ are temporal or environmentally driven interrupt conditions, $j = 1, \ldots, q$. The robot parses such words as $\dot{x} = f(x, k_{i_1}(t, x))$ until $\xi_{i_1}(t, x) = 1$, at which point the timer t is reset to 0, and $\dot{x} = f(x, k_{i_2}(t, x))$ until $\xi_{i_2}(t, x) = 1$, and so on. In other words, as in the case of the implementation of an execution string over regions of a partitioned environment discussed previously, the robot control strategy is a hybrid automaton (as illustrated at the "implementation level" of the hierarchy in Figure 1).

Having noted that promising results in this area have been obtained, a number of challenges remain largely unsolved, including: "Given a robot platform and a high-level mission, what is the minimal number of control modes guaranteeing that the mission is completed?" "Given a set of control modes, what is the set of overall behaviors achievable by the system?" "What is the best control program achievable from a set of control modes given a particular mission?" and "Given example trajectories, can the corresponding multi-modal control program be recovered?" In the next two sections, we highlight and discuss some of the steps that have been taken toward answering these questions.

Dealing with Nonholonomy and Control Constraints

Planning feasible trajectories for robots with motion constraints (e.g., a car that cannot move sideways) is a complex problem in which symbolic approaches were proved to achieve crucial simplifications. The intuition behind this approach is the following. Let us consider the problem of flying small-scale autonomous helicopters, making full use of their maneuvering capabilities. Dynamic models for such robotic vehicles are very complex, and not even well known away from hovering conditions. Yet, human pilots are able to control remotely such helicopters and perform extremely challenging aerobatic maneuvers with remarkable precision and skill. Arguably, pilots are not optimizing the helicopter trajectory and the control inputs in real-time—rather, they execute complex routines by sequencing well-rehearsed maneuvers. Using such a concept, it was possible to demonstrate completely automated acrobatic flight of small-scale helicopters [5].

The basic property enabling such a symbolic approach is the invariance of robot dynamics to certain transformations, such as translations and rotations about certain axis, and translations in time. This is a key characteristics of human-designed vehicles, since this ensures that motion-control skills learned at one time/location can be effectively used at other places and other times. Formally, this property is called symmetry and is associated to a certain group, such as the group of rigid body motion, or a subgroup, such as pure rotations and translations.

Control Quanta

The first example of symbolic control of a robotic system was introduced through the so-called control quanta, applicable to driftless systems with symmetries. (A driftless system is such that it stops when a zero control input is applied.) A control quantum is any (reasonable, e.g., measurable) control signal with finite time support. As such, it can also be seen as a symbol in a MDL, as described previously. The application of a control quantum to a driftless system results into a finite displacement of the system. Continuing with our helicopter example, let us consider that the control quantum q_i, $i = 1, \ldots, m$ makes the helicopter take off from $(0, 0)$ and land at $(\Delta x_i, \Delta y_i)$. After a sequence of n_1 instances of q_1, n_2 instances of q_2, etc., the helicopter will land at $\sum_{i=1}^{m} n_i(\Delta x_i, \Delta y_i)$. Note that, even though we are not using a detailed model of the dynamics of the helicopter, this claim is convincing since we expect the helicopter's dynamics to be invariant with respect to translations. It can be shown that a very small number of control quanta (three in this case) are sufficient to be able to move the helicopter arbitrarily close to any desired point on the ground. The right sequence of

control quanta achieving the desired displacement can be found in polynomial time, and does not rely on the knowledge of the detailed dynamics of the helicopter, but rather on the aggregate effect of each control quantum, as summarized in the vectors $(\Delta x_i, \Delta y_i)$.

Clearly, trajectories constructed in this way may not always be the most efficient: In our case, the execution of each symbol would require the helicopter to take off and land. A more sophisticated method for symbolic control is based on the notion of compatibility between control quanta. Let us consider for example a control quantum q_3 that makes the helicopter take off from the origin and hover at an altitude of 100 m, over a certain location $(\Delta x_3, \Delta y_3)$, and a control quantum q_4 that start from hover at 100 m, moves the helicopter by $(\Delta x_4, \Delta y_4)$. Finally, let q_5 be a control quantum that starts from hover at 100 m, over the origin, lands the helicopter at $(\Delta x_5, \Delta y_5)$. A new characteristics of these control quanta is the fact that they are not necessarily compatible with one another, i.e., they cannot be arbitrarily combined. For example, q_4 cannot be applied when the helicopter is on the ground; q_5 cannot be applied twice in a row; q_3 does not necessarily lead to an altitude of 200 m if repeated twice (e.g., because of ground effect). Note that q_1, q_2, and q_4 are compatible with themselves, i.e., they are repeatable. These rules for combining control quanta define a language, i.e., a subset of all the strings that can be obtained combining symbols from the alphabet $\{q_1, q_2, q_3, q_4, q_5\}$, that can be encoded as the set of strings accepted by a finite state machine like the one shown in Figure 3. For example, $(q_1, q_3, q_5, q_5, q_4, q_2, q_1)$ is a valid string, whereas (q_1, q_4, q_5) is not.

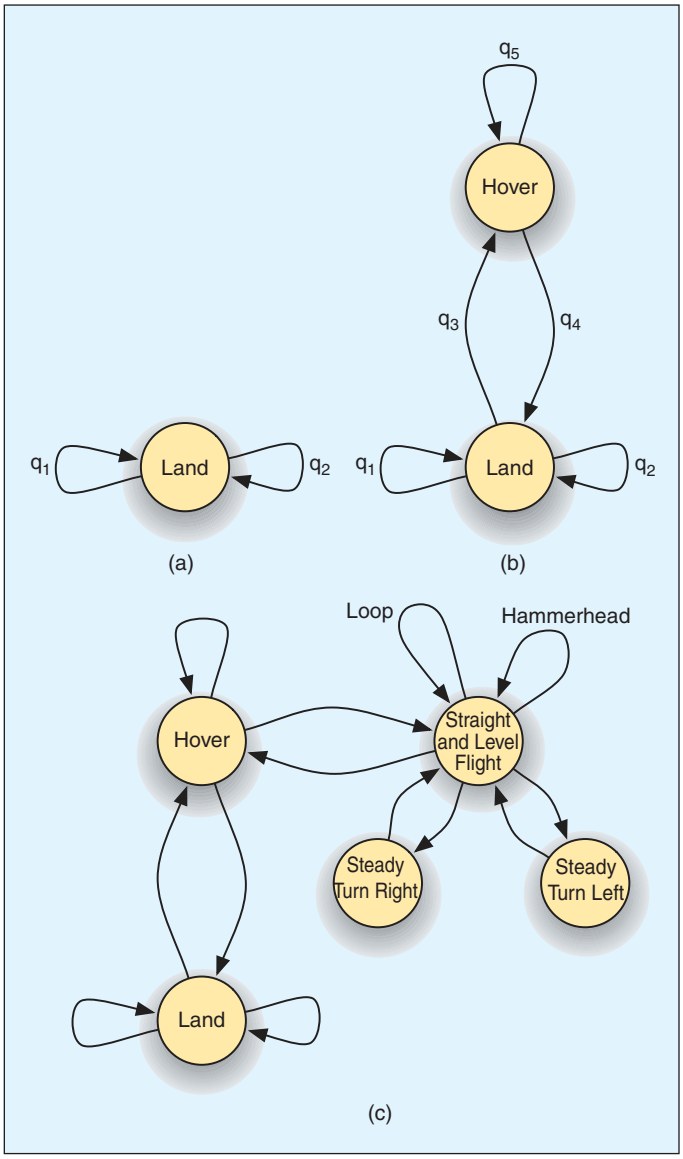


Figure 3. *A hierarchy of libraries for symbolic motion planning: (a) Control quanta with a common interface. (b) Control quanta with compatibility relations. (c) A maneuver automaton including several trim primitives, and acrobatic maneuvers.*

Motion Primitives

Instead of using control quanta chosen from a collection of controls, one could think of simplifying a robot control problem by piecing together, in an appropriate way, a set of elementary trajectories chosen from a small library—that are themselves guaranteed to satisfy the constraints. Such feasible trajectories that can be combined sequentially to produce more complicated trajectories are called *motion primitives*.

In [6], the following classes of motion primitives have been identified: i) trim primitives and ii) maneuvers. Trim primitives are repeatable primitives that can be arbitrarily cut, resulting in other repeatable primitives. This generalizes the notion of steady state, or relative equilibrium: Families of compatible trim primitives can be parameterized by the time duration, e.g., by a timer interrupt as in the previous. Maneuvers are motion primitives that are compatible with trim primitives. In other words, a maneuver is (the class of equivalence of) an arbitrary trajectory that starts and ends at steady-state conditions. For example, a loop is a maneuver that can only be started from steady and level flight at a given speed, and ends when steady and level flight is recovered. The language defined by the compatibility relations is encoded by a finite-state machine called a maneuver automaton, as depicted in Figure 3; such language allows the generation of motion plans combining steady-state trajectories with possibly complex and acrobatic maneuvers.

Motion primitives can be generated in several ways, for example by recording the actions of a human pilot; if an accurate model of the robot's dynamics is available, model-based approaches are also possible, e.g., to design optimal maneuvers. In real-world applications though, it is seldom the case that the application of a pre-recorded control input results in the same trajectory for the system. In order to reject unavoidable disturbances, open-loop motion primitives should be made robust using feedback control. Note that individual primitives can be allowed to be destabilizing, as long as closed paths on the automaton are stable: for example, acrobatic maneuvers can be performed in an essentially open-loop fashion, as long as a recovery period is allowed.

Feedback Encoding

Feedback can also be used to extend the applicability of symbolic motion planning strategies. In this more general case, the control quantum corresponding to a given symbol α also depends on the current state of the robot [i.e., $u = u(x, t, \alpha)$ in (1)]. This can essentially change the system at hand into a different system that is amenable to simpler motion control strategies. For example, in [7], the notion of feedback encoding is introduced, which affords the wealth of feedback-equivalence results in the nonlinear systems literature; in other words, through feedback encoding, broad classes of complex nonlinear systems can be abstracted (at least locally) to a linear system. Planning for such systems can then be achieved in a linear setting, and then projected back to the original systems by feedback decoding.

While the symbolic approach to motion planning described in this section has been applied successfully to challenging problems in autonomous mobile robotics, including acrobatic aircraft and off-road races, several challenges still need to be overcome: "What is the best choice of motion primitives for achieving a given class of tasks?" "Given an alphabet of motion primitives, what is the penalty associated with restricting the robot's trajectories to those obtained through combination of those primitives—with respect to a larger set of primitives?" and "Can we extend this symbolic approach to motion planning to multiple-robot systems?" Solving these problems will require a deep understanding of the interplay between the differential nature of the constraints on the dynamics of a robot, and the combinatorial nature of task specifications, discussed previously.

Low-Complexity, Symbolic Control Programs

One way of finding the appropriate set of control modes is by letting the control mode selection be driven by experimental data. For instance, one can envision a scenario in which a human operator is controlling a mobile platform, and then, through an analysis of the input-output sample paths, construct motion description languages that reproduce the human-driven robot behavior. Note that since the control modes are feedback laws, this would correspond to recovering control strategies rather than trajectories. If we let Σ denote the set of all recovered modes, a recovered mode-string $\sigma \in \Sigma^\star$ ($\Sigma^\star$ is the set of all finite length strings over Σ) requires a total of length$(\sigma)\log_2[\text{card}(\Sigma)]$ bits for its specification. This complexity measure (specification complexity) thus allows a tradeoff between the length [denoted length(σ)] of the string and the total number of available control modes [denoted card(Σ)].

In [8], a method was developed for recovering mode strings from empirical data with low specification complexity.

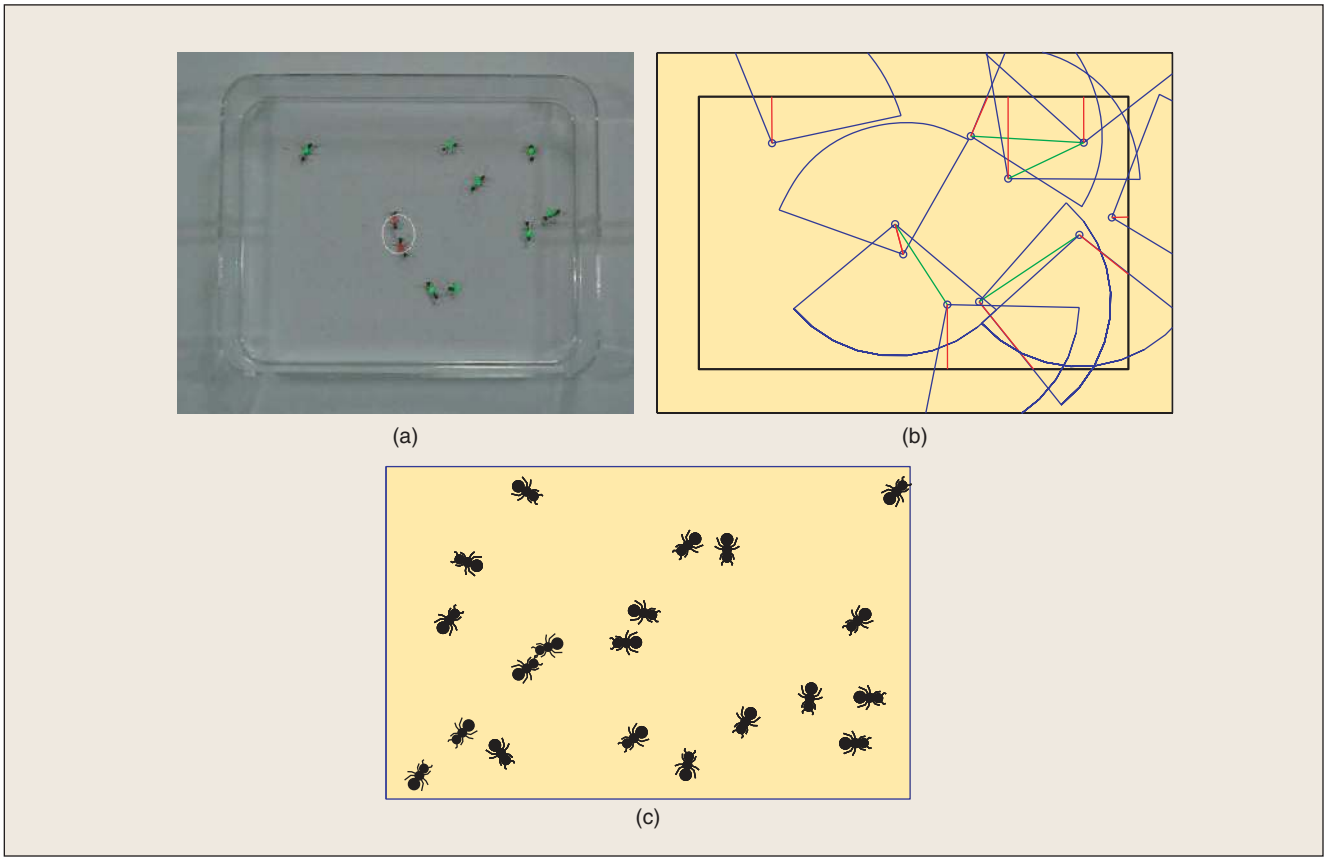


Figure 4. *Ten ants are moving around in a tank (a). The conical visual scope as well as the closest obstacles (dotted) and goals (dashed) for each individual ant (b). A simulation with 20 ants executing the recovered, low-complexity motion description languages (c).*

As an example, consider the problem of constructing mode sequences from data produced by a biological system, such as ten ants (Aphaenogaster cockerelli) moving in a tank with a camera mounted on top, as seen in Figure 4(a). Based on models of the ants' dynamics and perception regions, the video data can be translated into input-output strings, as seen in Figure 4(b). Following this, low-complexity mode strings can be recovered and redeployed on simulated ants with different initial conditions. A result of applying this methodology is shown in Figure 4(c).

As for the more philosophical question "What are the ants doing?" this method provides the answer in terms of mode strings rather than qualitative descriptions of ant behaviors. In fact, the driving motivation behind this line of inquiry is to come up with executable programs, i.e., strings of control laws and interrupt conditions that can be operated by robotic devices to produce ant-like behaviors rather than to produce insights into the lives of ants. However, a closer inspection of the recovered mode strings reveals that the most frequently occurring modes are qualitatively generating behaviors like "go straight slowly if no obstacles or goals are visible" or "go fast, turning left/right when an obstacle is to the right/left and/or the goal is to the left/right."

Now, given that we have a measure of complexity as well as a method for constructing the mode set, one can ask whether or not the recovered mode set is rich enough for the set of robotic missions that one might wish to undertake. This relates to the classic tradeoff between complexity and expressiveness. Given that the underlying objective of the multimodal control design is to generate as rich a set of trajectories as possible, one can tie the issue of expressiveness directly to an optimal control problem. Suppose for example that the behavior one wants the system to exhibit can be characterized by $\tilde{x}(t)$, while the state of the system is $x^{\sigma}(t)$. Then, given a mode set Σ, one can define the tracking error as $J(\sigma, \tilde{x}) = \|x^{\sigma} - \tilde{x}\|$ (in some appropriate functional norm), with $\sigma \in \Sigma^{\star}$.

The problem of minimizing J with respect to σ can be (locally) solved using tools from hybrid optimal control. This problem can moreover be extended in order to investigate the expressiveness ($\mathcal{E}$) of the mode set by solving the differential game, max-min problem

$$\mathcal{E} = \max_{\tilde{x} \in \tilde{X}} \left\{ \min_{\sigma \in \Sigma^{\star}} \{J(\sigma, \tilde{x})\} \right\},$$

where $\tilde{x}$ encodes the set of all desired trajectories. Unfortunately, solving this problem is by no means simple, and as such, effective computational methods for obtaining the expressiveness of a given mode set remains one of the many challenges in the subarea of symbolic control that we have termed control-driven discretizations.

Multirobot systems

In multiagent mobile systems, it is believed that local communication and control strategies inspired from natural systems such as flocks of birds or schools of fish can lead to useful and predictable global behaviors. Alternatively, such communication and control protocols can be achieved through the use of embedded graph grammars [9], which we briefly review in the rest of this section.

In this formalism, robots are represented by vertices in a graph. The edges in the graph represent coordination or communication. Labels on the vertices or edges represent internal states, which can be used, for example, by the communication protocol. A grammatical rule is a rewrite schema of the form $L \rightharpoonup R$ where L and R are small graphs. We interpret a rule as follows: If a subgraph of the global state graph matches L, we update that subgraph to match R, thereby arriving at a

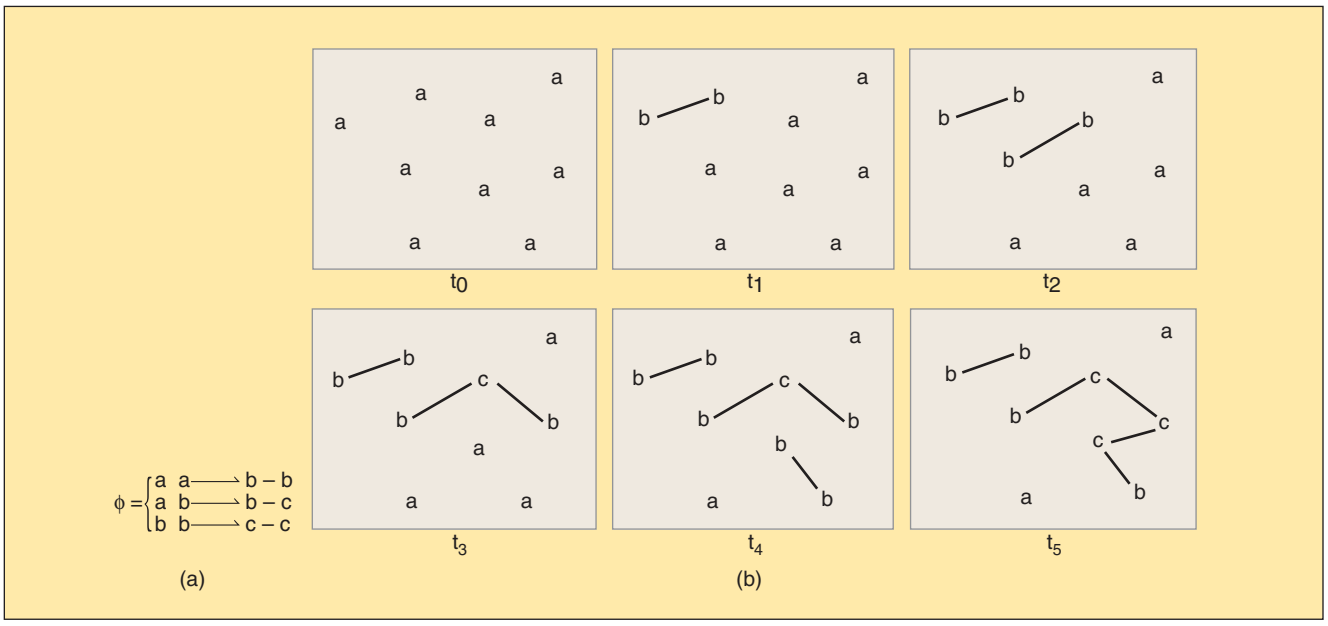


Figure 5. *(a) Simple graph grammar. (b) Sample trajectory admitted by the graph grammar.*

(nondeterministic) sequence of graphs representing a trajectory: $G_0 \to G_1 \to G_2 \to G_3 \ldots$ Figure 5 shows an example.

To associate motion controllers with each robot requires more machinery. First, we associate a workspace position to each vertex in the graph that denotes the position of the corresponding robot (i.e., we embed the graph into the workspace). Second, we associate a continuous guard to each grammatical rule that states what condition on, for example, the locations of the robots must hold for the rule to be applicable. Third, we associate a motion controller with each symbol. This results in an embedded graph grammar (EGG) [10]. An EGG rule essentially allows for statements like "If there are robots i, j and k in the embedded graph such that $||x_i - x_k|| \approx r$ and the communication subgraph for i, j and k matches the left hand side of the rule, then change the subgraph according to the right-hand side."

Graph grammars have been implemented directly with a variety of distributed robot systems. The advantage of the approach is that each robot in the network can simply be a graph grammar interpreter. Low level communication protocols can be separated from the task definition (via a graph grammar) by a layer of abstraction. This has also been accomplished with a stochastic version of graph grammars where rules can be thought of as programmable chemical reactions. In both of these cases, the level of abstraction afforded by the graph grammar view has enabled straightforward specification and implementation of complex multivehicle tasks.

Conclusion

We have illustrated different research trends that use symbolic techniques for robot motion planning and control. As it often happens in new research areas, contributions to this topic started at about the same time by different groups with different emphasis, approaches, and notation. In this article, we tried to describe a framework in which many of the current methods and ideas can be placed and to provide a coherent picture of what we want to do, what have we got so far, and what the main missing pieces are. Generally speaking, the aim of symbolic control as we envision it is to enable the usage of methods of formal logic, languages, and automata theory for solving effectively complex planning problems for robots and teams of robots.

An example of the grand challenges we have in mind is to enable a naive user to specify the daily chores of her/his domestic robotic appliances in plain natural language—tidy up rooms in the house daily and collect garments to do the laundry every Tuesday, only after kids have left for school—and automatically translate this in device-independent programs, to be downloaded and executed on robots, so as to obtain a dependable performance of the task in spite of the complexity and unpredictability of the environment.

The results presented in this article can be divided in two groups: top-down approaches, whereby formal logic tools are employed on rather abstract models of robots; and bottom up approaches, whose aim is to provide means by which such abstractions are possible and effective. The two ends do not quite tie as yet, and much work remains to be done in both directions to obtain generally applicable methods. However, the prospects of symbolic control of robots are definitely promising, and the challenging nature of problems to be solved warrants for the interest of a wide community of researchers.

Even though automatic control of robots and teams of robots from high level specifications given as formulas of some temporal logic is useful and possible, several fundamental questions remain to be answered.

Acknowledgment

The authors would like to thank the audience of the workshop on "Symbolic Approaches for Robot Motion Planning and Control" (held as part of Robotics: Science and Systems, 2006) for their active participation and challenging questions, which helped focus the main ideas of this article.

Keywords

Formal methods, motion planning, robot control, self-assembly.

References

[1] C. Belta, V. Isler, and G.J. Pappas, "Discrete abstractions for robot planning and control in polygonal environments," *IEEE Trans. Robot.*, vol. 21, no. 5, pp. 864–874, Oct. 2005.

[2] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from LTL specifications," in *Hybrid Systems: Computation and Control: 9th International Workshop*, ser. Lecture Notes in Computer Science, J. Hespanha and A. Tiwari, Eds. Berlin, Germany: Springer-Verlgag, vol. 3927, pp. 333–347, 2006.

[3] G. Fainekos, S. Loizou, and G.J. Pappas, "Translating temporal logic to controller specifications," in *Proc. 45th IEEE Conf. Decision Control*, San Diego, CA, Dec. 2006, pp. 899–904.

[4] M. Egerstedt and R. Brockett, "Feedback can reduce the specificat complexity of motor programs," *IEEE Trans. Autom. Control*, vol. 48, no. 2, pp. 213–223, Feb. 2003.

[5] V. Gavrilets, B. Mettler, and E. Feron, "Human-inspired control logic for automated maneuvering of miniature helicopter," *AIAA J. Guid., Control, Dyna.*, vol. 27, no. 5, pp. 752–759, 2004.

[6] E. Frazzoli, M.A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Trans. Robot.*, vol. 21, no. 6, pp. 1077–1091, Dec. 2005.

[7] A. Bicchi, A. Marigo, and B. Piccoli, "Feedback encoding for efficient symbolic control of dynamical systems," *IEEE Trans. Autom. Control*, vol. 51, no. 1, pp. 1–16, Jan. 2006.

[8] F. Delmotte, T. Mehta, and M. Egerstedt, "Modebox a software tool for obtaining hybrid control strategies from data," *IEEE Robot. Automat. Mag.*, to be published.

[9] E. Klavins, R. Ghrist, and D. Lipsky, "A grammatical approach to self-organizing robotic systems," *IEEE Trans. Autom. Control*, vol. 51, no. 6, pp. 949–962, Jun. 2006.

[10] J. McNew and E. Klavins, "Locally interacting hybrid systems with embedded graph grammars," in *Proc. IEEE Conf. Decision Control*, San Diego, CA, 2006, pp. 6080–6087.

Calin Belta received the M.Sc. and Ph.D. degrees in mechanical engineering from the University of Pennsylvania, Philadelphia, in 2003. He is currently an Assistant Professor in the College of Engineering at Boston University, Boston, MA. His research interests include verification and control of hybrid systems, robot planning and control, gene and metabolic networks. Dr. Belta received an NSF CAREER award in 2005, a Fulbright study award in 1997, and was the Valedictorian of his class in 1995. He received the best poster award at the International Conference on Systems Biology in 2004, and was a finalist for both the ASME Design Engineering Technical Conference best paper award in 2002 and for the Anton Philips best student paper award at the IEEE International Conference on Robotics and Automation in 2001.

Antonio Bicchi is a Professor of Automatic Control and Robotics at the University of Pisa. He graduated from the University of Bologna, Bologna, Italy, in 1988, and was a postdoctoral scholar at the Massachusetts Institute of Technology A.I. Laboratory, Cambridge, between 1988–1990. His main research interests are in dynamics, kinematics, and control of complex mechanical systems, including robots, autonomous vehicles, and automotive systems, haptics and dextrous manipulation; and theory and control of nonlinear systems, in particular hybrid systems. He has published more than 200 papers in international journals, books, and refereed conferences. He currently serves as the Director of the Interdepartmental Research Center "E. Piaggio" of the University of Pisa, Pisa, Italy. Dr. Bicchi is an IEEE Fellow and Vice President of the IEEE Robotics and Automation Society for Member Activities. He has served as an IEEE-RAS Distinguished Lecturer, an Editor for several scientific journals, and a Chairman of conferences such as WorldHaptics (2005) and Hybrid Systems: Computation and Control (2007).

Magnus Egerstedt is an Associate Professor in the School of Electrical and Computer Engineering at the Georgia Institute of Technology (Georgia Tech), Atlanta. He received the M.S. degree in engineering physics and the Ph.D. degree in applied mathematics from the Royal Institute of Technology, Stockholm, Sweden, in 1996 and 2000, respectively. His research interests include optimal control as well as modeling and analysis of hybrid and discrete-event systems, with emphasis on motion planning and control of (teams of) mobile robots, and he has authored over 100 papers in the areas of robotics and controls. He is the Chair of the Systems and Controls Technical Interest Group at the School of Electrical and Computer Engineering at Georgia Tech. Dr. Egerstedt is a Senior Member of the IEEE. He received the ECE/GT Outstanding Junior Faculty Member Award in 2005 and the CAREER award from the U.S. National Science Foundation in 2003.

Emilio Frazzoli is an Assistant Professor of Aeronautics and Astronautics at the Massachusetts Institute of Technology (MIT(, Cambridge. He received the Laurea degree in aerospace engineering from the University of Rome, "La Sapienza," Rome, Italy, in 1994, and the Ph. D. degree in navigation and control systems from the Department of Aeronautics and Astronautics, MIT, in 2001. From 2001 to 2004, he was an Assistant Professor of Aerospace Engineering at the University of Illinois at Urbana-Champaign. From 2004 to 2006, he was an Assistant Professor of Mechanical and Aerospace Engineering at the University of California, Los Angeles. He was the recipient of an NSF CAREER award in 2002. His current research interests include algorithmic, computational, and geometric approaches to the design and development of decision and control architectures for complex networked and autonomous systems.

Eric Klavins is an Assistant Professor of Electrical Engineering at the University of Washington, Seattle. He received the B.S. degree in computer science in 1996 from San Francisco State University, San Francisco, CA, and the M.S. and Ph.D. degrees in computer science and engineering in 1999 and 2001 from the University of Michigan, Ann Arbor, MI. From 2001 to 2003, he was a Postdoctoral Scholar in the Control and Dynamical Systems Department, the California Institute of Technology, Pasadena. As a graduate student, he was supported by a Charles DeVlieg Fellowship for Manufacturing. In 2001, he received an NSF CAREER award entitled "Programmable Robotic Self-Assembly." His research interests include cooperative control, robotics, distributed systems, concurrency, self-organizing systems, and nanotechnology.

George J. Pappas is an Associate Professor in the Department of Electrical and Systems Engineering at the University of Pennsylvania, Philadelphia. He also holds a secondary appointment in the Departments of Computer and Information Sciences, and Mechanical Engineering and Applied Mechanics, and is the Director of the GRASP Lab. His research focuses on the interface of control theory with computer science and in particular, hybrid systems, embedded systems, and hierarchical and distributed control systems, with applications to unmanned aerial vehicles, flight management systems, distributed robotics, and biomolecular networks. He has published over 100 publications and has won various awards, including the National Science Foundation Presidential Early Career Award for Science and Engineering (PECASE).

Address for Correspondence: Calin Belta, Boston University, 15 Saint Mary's St., Room EMB147, Brookline, MA 02446 USA. Phone: +1 617 353 9586. Fax: +1 617 353 5548. E-mail: cbelta@bu.edu.