# Synthesis of Hybrid Constraint-Based Controllers

Ying Zhang[*] and Alan K. Mackworth[**]

Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada, V6T 1Z4
E-mail: zhang,mack@cs.ubc.ca

**Abstract.** A robot is an integrated system, with a controller embedded in its plant. We take a robotic system to be the coupling of a robot to its environment. Robotic systems are, in general, hybrid dynamic systems, consisting of continuous, discrete and event-driven components. We call the dynamic relationship of a robot and its environment the behavior of the robotic system. The problem of control synthesis is: given a requirements specification for the behavior, and given dynamic models of the plant and the environment, generate a controller so that the behavior of the robotic system satisfies the specification. We have developed a formal language, Timed Linear Temporal Logic (TLTL) [17], for requirements specification. We have also developed a semantic model, Constraint Nets [19], for modeling hybrid dynamic systems. In this paper, we study the problem of control synthesis using these representations. Control synthesis in general is difficult. We first focus on a special class of requirements specification, called constraint-based specification, in which constraints are associated with properties such as safety, reachability and persistence. Then we develop a systematic approach to synthesizing controllers using constraint methods, in which controllers are embedded constraint solvers that solve constraints in real-time. Finally, we consider hierarchical control structures, in which the higher levels embody digital/symbolic event-driven control derived from discrete constraint methods and the lower levels incorporate analog control based on continuous constraint methods. We illustrate these techniques using a robot soccer player as a running example.

## 1 Control Synthesis
*Where does the problem fit in?*

Robots are generally composed of electromechanical parts with multiple sensors and actuators. Robots should be reactive as well as purposive systems, closely coupled with their environments; they must deal with inconsistent, incomplete, unreliable and delayed information from various sources. Robots are usually complex, hierarchically organized and physically distributed; each component functions according to its own

[*] Current address: The Wilson Center for Technology, Xerox Corporation, 800-Phillips Road, M/S 128-51E, Webster, N.Y., U.S.A., 14580. E-mail: zhang@wrc.xerox.com
[**] Fellow, Canadian Institute for Advanced Research.

dynamics. The overall behavior of a robot emerges from coordination among its various parts and interaction with its environment. We call the coupling of a robot and its environment a robotic system, and the dynamic relationship of a robot and its environment the behavior of the robotic system.

A robot controller (or control system) is a subsystem of a robot, designed to regulate its behavior to meet certain requirements. In general, a robot controller is an integrated software/hardware system implemented on various digital/analog devices. Thus, from the systemic point of view, a robotic system consists of a controller, a plant and an environment as shown in Fig. 1, where $X$ is a set of plant variables, $U$ is the set of control variables, and $Y$ is a set of environment variables. Some of the plant and environment variables may not be observable by the controller.

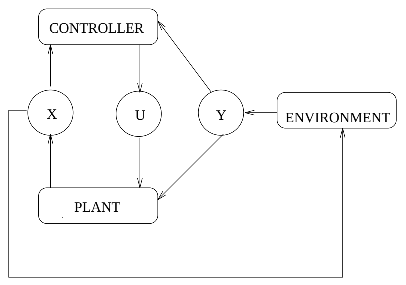

**Fig. 1.** A robotic system

Robotic systems are typical hybrid systems, consisting of continuous, discrete and event-driven components. In order to develop a robotic system, analyze its behavior and understand its underlying physics, we have developed a semantic model for hybrid dynamic systems, that we call Constraint Nets (CN). Using this model, we can characterize the components of a system and derive the behavior of the overall system.

Control systems are designed to meet certain requirements. Typical requirements include safety, reachability and persistence. Safety declares that a system should never be in a certain situation. Reachability declares that a system should reach a certain goal eventually. Persistence declares that a system should achieve a certain goal infinitely often. In order to specify requirements formally, we have developed two languages, Timed Linear Temporal Logic (TLTL) and timed $\forall$-automata. In this paper, we will introduce only TLTL.

The problem of control synthesis is stated as follows: given a requirements specification, and the dynamic models of the plant and the environment, produce (a dynamic model of) a controller such that the behavior of the robotic system meets the given requirements. Control synthesis in general is difficult. As a first step, we focus on a special class of requirements specification — constraint-based specification, in which constraints are associated with properties such as safety, reachability and persistence. We then develop a systematic approach to control synthesis from constraint-based specifications, in which controllers are constraint solvers that solve constraints in real-time.

Fig. 2 presents an overall framework of our approach to the design and analysis of robotic systems, indicating where the problem of control synthesis fits in. In this framework, control synthesis and behavior verification are coupled together to generate a "correct" system. In this paper, we will focus only on control synthesis.

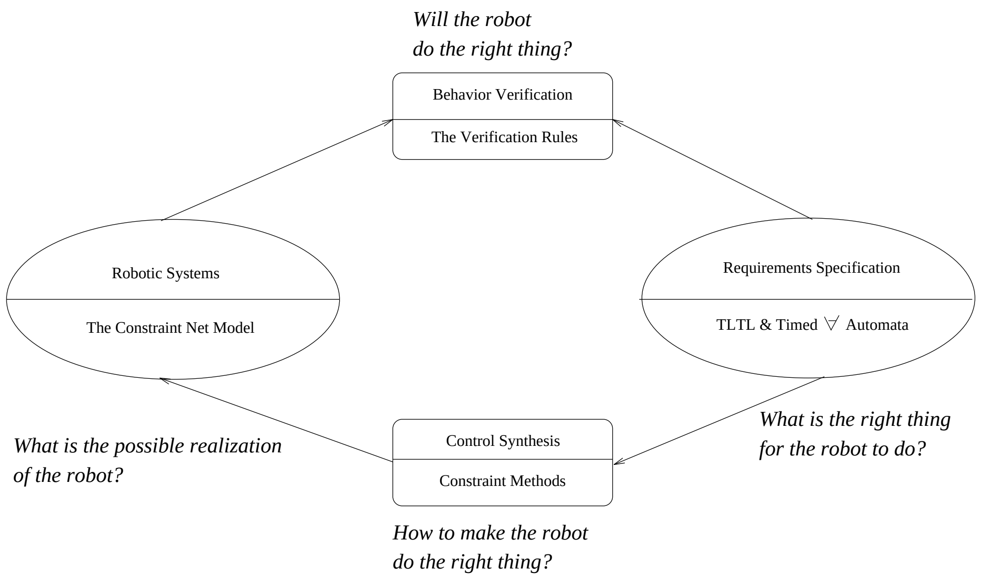
Will the robot do the right thing?
Behavior Verification
The Verification Rules
Robotic Systems
The Constraint Net Model
Requirements Specification
TLTL & Timed ∀ Automata
What is the possible realization of the robot?
Control Synthesis
Constraint Methods
What is the right thing for the robot to do?
How to make the robot do the right thing?

**Fig. 2.** Where does control synthesis fit in?

Let us introduce an example that will be used throughout this paper. In our Laboratory for Computational Intelligence, a testbed has been developed for radio-controlled cars playing soccer using visual feedback [15]. Each "soccer player" has a car-like mobile base. It can move forward and backward with a throttle setting, and can make turns by steering its two front wheels. However, it cannot move sideways and its turns are limited by mechanical stops in the steering gear.

Fig. 3 illustrates the configuration of a car. Let $v$ be the velocity of the car and $\alpha$ be the current steering angle of the front wheels; $v$ and $\alpha$ can be considered as the control inputs to the car. The dynamics of the car can be simply modeled by the following differential equations [7]:

$$\dot{x} = v\cos(\theta), \quad \dot{y} = v\sin(\theta), \quad \dot{\theta} = v/R \tag{1}$$

where $(x, y)$ is the position of the tail of the car, $\theta$ is the heading and $R = L/\tan(\alpha)$ is the turning radius given the length of the car $L$. The environment of this system is a moving ball, a target (the goal) and a field with boundaries. A requirements specification for the robot is to kick or dribble the ball to the target repeatedly. The

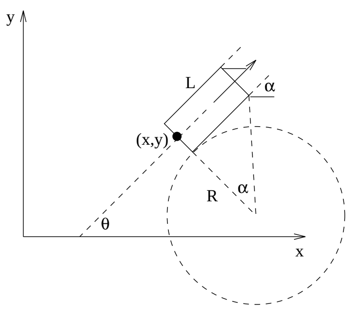

**Fig. 3.** The configuration of a car

controller of the car is equipped with both digital and analog devices [15]. How can we synthesize such a controller?

The rest of this paper is organized as follows. Section 2 briefly introduces the Constraint Net model. Section 3 presents TLTL. Section 4 focuses on constraint-based specification. Section 5 develops constraint-based control. Section 6 discusses general control structures. Finally, Section 7 concludes the paper and proposes further research on control synthesis.

## 2 Constraint Nets

*What is synthesized?*

Constraint Nets (CN) is a semantic model for hybrid dynamic systems. CN is an abstraction and generalization of dataflow networks [8, 2, 4, 3]. Any (causal) system with discrete/continuous time, discrete/continuous (state) variables, and asynchronous/synchronous event structures can be modeled. Furthermore, a system can be modeled hierarchically using aggregation operators; the dynamics of the environment as well as the dynamics of the plant and the controller can be modeled individually and then integrated.

Our approach to developing this model for hybrid systems is motivated by the following arguments. First, hybrid systems consist of interacting discrete and continuous components. Instead of fixing a model with particular time and domain structures, a model for hybrid systems should be developed on both abstract time structures and abstract data types. Second, hybrid systems are complex systems with multiple components. A model for hybrid systems should support hierarchy and modularity. Third, hybrid systems are generalizations of basic discrete or continuous systems. A model for hybrid systems should be at least as powerful as existing models of computation and control. In short, a model for hybrid systems should be unitary, modular, and powerful.

In the rest of this section, we briefly introduce some concepts of dynamic systems, and the syntax and semantics of CN. For a comprehensive introduction to CN, the reader is referred to [19] or [17].

## 2.1 Basic concepts

Let $\mathcal{R}$ be the set of real numbers and $\mathcal{R}^+$ be the set of nonnegative real numbers. A *metric* on a set $X$ is a function $d : X \times X \to \mathcal{R}^+$ satisfying the following axioms for all $x, y, z \in X$:

1. $d(x, y) = d(y, x)$.
2. $d(x, y) + d(y, z) \geq d(x, z)$.
3. $d(x, y) = 0$ iff $x = y$.

A *metric space* is a pair $\langle X, d \rangle$ where $X$ is a set and $d$ is a metric on $X$. In a metric space $\langle X, d \rangle$, $d(x, y)$ is called "the distance between $x$ and $y$." We will use $X$ to denote the metric space $\langle X, d \rangle$ if no ambiguity arises.

Following are some basic concepts of dynamic systems developed in [17].

- *Time* can be abstracted as a linearly ordered set $\langle \mathcal{T}, \leq \rangle$ with a least element $\mathbf{0}$ as the start time point, and a metric $d$ on set $\mathcal{T}$. Using this abstract notion, time can be either discrete, continuous or event-based.
- A *domain* can be either simple or composite. A simple domain denotes a simple data type, such as reals, integers, Booleans and characters; a composite domain denotes a structured data type, such as arrays, vectors, strings, objects, structures and records. Formally, a *simple domain* is a pair $\langle A \cup \{\perp_A\}, d_A \rangle$ where $A$ is a set, $\perp_A \notin A$ means undefined in $A$, and $d_A$ is a metric on $A$. A *composite domain* is a product of simple domains. Using this abstract notion, a domain can be numerical or symbolic, discrete or continuous.
- A trace characterizes the change of values of a variable over time. Formally, a *trace* is a mapping $v : \mathcal{T} \to A$ from time $\mathcal{T}$ to domain $A$. An *event trace* is a special type of trace whose domain is Boolean. An *event* in an event trace is a transition from 0 to 1 or from 1 to 0. An event trace characterizes some event-based time where the set of events in the trace is the time set.
- A *transduction* is a mapping from a tuple of input traces to an output trace that satisfies the causality constraint between the inputs and the output, i.e., the output value at any time depends only on its inputs up to that time. For instance, a state automaton with an initial state defines a transduction on discrete time; a temporal integration with a given initial value is a typical transduction on continuous time. Just as nullary functions represent constants, nullary transductions represent traces.

  We characterize two classes of atomic transductions: simple transductions and simple event-driven transductions.
- A *simple transduction* is a transliteration or a delay.
- A *transliteration* is a pointwise extension of a function. Intuitively, a transliteration is a transformational process without memory (internal state), such as a combinational circuit.
- A *delay* transduction is a sequential process where the output value at any time is the input value at a previous time. Discrete-time state transitions and continuous-time integrations can be modeled using delays.
- A simple *event-driven transduction* is a simple transduction augmented with an extra input which is an event trace; the transduction operates at every event and its output value holds between two events.

## 2.2 The Constraint Net model

A constraint net consists of a finite set of locations, a finite set of transductions and a finite set of connections. Formally, a *constraint net* is a triple $CN = \langle Lc, Td, Cn\rangle$, where $Lc$ is a finite set of *locations*, $Td$ is a finite set of labels of *transductions*, each with an *output port* and a set of *input ports*, $Cn$ is a set of *connections* between locations and ports, with the following restrictions: (1) there is at most one output port connected to each location, (2) each port of a transduction connects to a unique location and (3) no location is isolated. A location can be regarded as a wire, a channel, a variable, or a memory cell. Each transduction is a causal mapping from inputs to outputs over time, operating according to a certain reference time or activated by external events.

A location is an *output* of the constraint net if it is connected to the output of some transduction; it is otherwise an *input*. A constraint net is *open* if there is an input location; it is otherwise *closed*.

Semantically, a constraint net represents a set of equations, with locations as variables and transductions as functions. The *semantics* of the constraint net, with each location denoting a trace, is the least solution of the set of equations [19, 17].

A complex system is generally composed of multiple components. We define a *module* as a constraint net with a set of locations as its interface. A constraint net can be composed hierarchically using modular and aggregation operators on modules. The semantics of a system can be obtained hierarchically from the semantics of its subsystems and their connections.

Given $CN$, a constraint net model of a dynamic system, the abstract behavior of the system is the semantics of $CN$, denoted $[\![CN]\!]$, i.e., the set of input/output traces satisfying the model.

A constraint net is depicted by a bipartite graph where locations are depicted by circles, transductions by boxes and connections by arcs. A module is depicted by a box with rounded corners. For example, the dynamics of the car, characterized by Eq. (1), is denoted by an open constraint net, as shown in Fig. 4 where *sin*, *cos*, *tan* and $*$ are transliterations, and $\int$ is a temporal integrator. The car module is defined by choosing locations $v, \alpha, x, y, \theta$ as the interface.

We can model a control system as a module that may be further decomposed into a hierarchy of interactive modules. The higher levels are composed of event-driven transductions and the lower levels are analog control components. Furthermore, the environment of the robot can be modeled as a module as well. A robotic system can be modeled as an integration of a plant, a controller and an environment (Fig. 1). The semantics (or behavior) of the system is the solution of the following equations:

$$X = PLANT(U, Y), \tag{2}$$

$$U = CONTROLLER(X, Y), \tag{3}$$

$$Y = ENVIRONMENT(X). \tag{4}$$

Note that $PLANT$, $CONTROLLER$ and $ENVIRONMENT$ are transductions (not simple functions), and the solution gives $X$, $Y$ and $U$ as tuples of traces (not values).

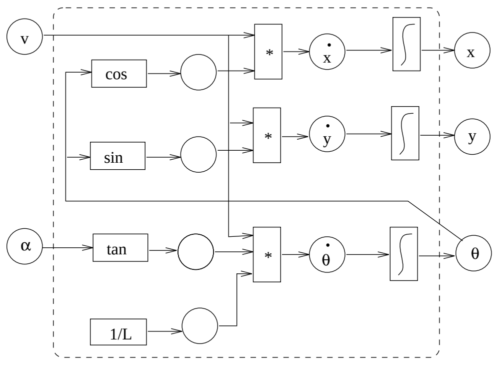

**Fig. 4.** The constraint net for the dynamics of the car, Eq. (1)

As we can see here, a robot, composed of a plant and a controller, is an open system, and a robotic system, composed of a robot and its environment, is a closed system.

## 3 Timed Linear Temporal Logic Specification *What is required?*

A model of a dynamic system represents the whole system as a set of components and their connections. However, the behavior of the system is not explicitly represented, since most dynamic systems have no closed form solutions.

Most design requirements can be expressed by qualitative properties and can be satisfied by many models (implementations). A *requirements specification* $\mathcal{R}$ for a system $CN$ is a set of allowable input/output traces of the system: the behavior of $CN$ satisfies its requirements specification $\mathcal{R}$, written $[\![CN]\!] \models \mathcal{R}$, iff $[\![CN]\!] \subseteq \mathcal{R}$. For example, $CN : x = \int(x_0)(-x)$ is a model of dynamic system whose behavior, $x(t) = x_0 e^{-t}$, satisfies the requirements specification $\mathcal{R} : \lim_{t\to\infty} x(t) = 0$.

The problem of control synthesis can be formalized as follows: given a requirements specification $\mathcal{R}$, the model of the plant $PLANT$ and the model of the environment $ENVIRONMENT$, synthesize a model of the controller $CONTROLLER$, such that $[\![CN]\!] \models \mathcal{R}$ where $CN$ is modeled by Eqs. (2), (3) and (4).

We have developed Timed Linear Temporal Logic (TLTL) and timed ∀-automata, generalized from Linear Temporal Logics [11] and ∀-automata [10], for specifying requirements. In this paper, we introduce only TLTL.

The basic form of the propositional linear temporal logic (PLTL) is the classical propositional logic extended with temporal operators. Formally, the syntax of the logic is defined as follows. Let $\Phi$ be a set of propositions. The basic syntax is defined in BNF:

$$F ::= \mathit{false} \mid p \mid F_1 \rightarrow F_2 \mid F_1 \,\mathcal{U}\, F_2 \mid F_1 \,\mathcal{S}\, F_2$$

where $p \in \Phi$ is a proposition, $\rightarrow$ is a logical connective denoting “implication,” $\mathcal{U}$ is a temporal operator denoting “until” and $\mathcal{S}$ is a temporal operator denoting “since.”

Formally, the semantics of the logic is defined as follows. Let $\mathcal{T}$ and $A$ be time and domain, respectively, and $v : \mathcal{T} \longrightarrow A$ be a trace. Note that $\mathcal{T}$ and $A$ can be either discrete or continuous. Let $V : \Phi \longrightarrow 2^A$ be an interpretation that assigns to each proposition $p \in \Phi$ a subset $V(p)$ of $A$. We will use $a \models p$ to denote $a \in V(p)$. Then $v \models_t F$ denotes that trace $v$ satisfies formula $F$ at time $t$:

- $v \not\models_t false$.
- $v \models_t p$ for $p \in \Phi$ iff $v(t) \models p$.
- $v \models_t F_1 \longrightarrow F_2$ iff $v \models_t F_1$ implies $v \models_t F_2$.
- $v \models_t F_1 \mathcal{U} F_2$ iff $\exists t' > t, v \models_{t'} F_2$ and $\forall t'', t < t'' < t', v \models_{t''} F_1$.
- $v \models_t F_1 \mathcal{S} F_2$ iff $\exists t' < t, v \models_{t'} F_2$ and $\forall t'', t' < t'' < t, v \models_{t''} F_1$.

We will use $v \models F$ to denote that $v$ satisfies $F$ initially, i.e., $v \models_{\mathbf{0}} F$.

More logical connectives and temporal operators can be defined using the basic logic connective $\longrightarrow$ and the basic temporal operators $\mathcal{U}$ and $\mathcal{S}$.

Some commonly used logical connectives are defined as follows:

- *Negation*: $\neg F \equiv F \longrightarrow false$.
- *True*: $true \equiv \neg false$.
- *Disjunction*: $F_1 \vee F_2 \equiv \neg F_1 \longrightarrow F_2$.
- *Conjunction*: $F_1 \wedge F_2 \equiv \neg(F_1 \longrightarrow \neg F_2)$.

Some commonly used temporal operators are defined as follows:

- *Eventually*: $\Diamond F \equiv F \vee true \mathcal{U} F$.
- *Always*: $\Box F \equiv \neg \Diamond \neg F$.
- *Next*: $\bigcirc F \equiv F \mathcal{U} F$.
- *Previous*: $\ominus F \equiv F \mathcal{S} F$.

Various stronger and weaker variations of these temporal operators [5] can also be defined.

We define some more abbreviations that are convenient to use in many situations.

- $final \equiv \neg \bigcirc true$.
- $initial \equiv \neg \ominus true$.
- $rise(p) \equiv (\neg p \wedge \bigcirc p) \vee (\ominus \neg p \wedge p)$.
- $change(p) \equiv rise(p) \vee rise(\neg p)$.
- $event(p) \equiv (\ominus \neg p \wedge p) \vee (\ominus p \wedge \neg p)$.

Some important properties of behaviors can be specified using PLTL.

- *Safety*: If $B$ is a proposition denoting a bad situation, $\Box \neg B$.
- *Reachability*: If $G$ is a proposition denoting a final goal, $\Diamond \Box G$.
- *Persistence*: If $P$ is a proposition denoting a persistent condition, $\Box \Diamond P$.

In order to specify the metric properties of time, we develop Timed Linear Temporal Logic (TLTL). The basic syntax and semantics of TLTL are the same as those of PLTL. In addition, we augment the basic form of PLTL with two real-time operators. Let $\tau > 0$ be a positive real number, $T_{t+\tau} = \{t' | t < t', d(t, t') \leq \tau\}$ and $T_{t-\tau} = \{t' | t' < t, d(t', t) \leq \tau\}$. Two real-time operators are defined as follows:

- $v \models_t F_1 \mathcal{U}^\tau F_2$ iff $\exists t' \in T_{t+\tau}$, $v \models_{t'} F_2$ and $\forall t'', t < t'' < t', v \models_{t''} F_1$.
- $v \models_t F_1 \mathcal{S}^\tau F_2$ iff $\exists t' \in T_{t-\tau}$, $v \models_{t'} F_2$ and $\forall t'', t' < t'' < t, v \models_{t''} F_1$.

Other real-time and temporal operators can be defined using the two basic real-time operators.

- $\Diamond^\tau F \equiv true\,\mathcal{U}^\tau F$.
- $\Box^\tau F \equiv \neg(\Diamond^\tau \neg F)$.
- $\Diamond_\tau F \equiv true\,\mathcal{S}^\tau F$.
- $\Box_\tau F \equiv \neg(\Diamond_\tau \neg F)$.

With real-time operators, real-time properties can be specified, for example, real-time response can be specified as $\Box(E \rightarrow \Diamond^\tau R)$.

In our framework the control synthesis problem is stated as follows: given a set of TLTL formulas, and given constraint net models of the plant and the environment, produce a constraint net model of the controller so that the behavior of the overall system satisfies the given requirements. In its full generality this problem is too difficult. As a first step, we focus on a special class of requirements and develop a systematic, but not automatic, approach to control synthesis.

## 4 Constraint-Based Specification

*What should be satisfied?*

We consider constraints as relations on a set of state variables; the solution set of the constraints consists of the state variable tuples that satisfy all the constraints. We call the behavior of a dynamic system constraint-based if the system is asymptotically stable at the solution set of the given constraints, i.e., whenever the system diverges because of some disturbance, it will eventually return to the set satisfying the constraints.

Most robotic systems are constraint-based, where the constraints may include physical limitations, environmental restrictions, and safety and goal requirements. Most learning and adaptive dynamic systems exhibit some forms of constraint-based behaviors as well.

A constraint-based specification characterizes a desired constraint-based behavior. Formally, let $C$ be a set of constraints and $sol(C)$ be the solution set of $C$, and let $C^\epsilon$ be a neighborhood of $sol(C)$; $C^\epsilon$ and $sol(C)$ can be taken as propositions whose interpretations are themselves. TLTL formulas $\Box C^\epsilon$, $\Diamond\Box C^\epsilon$ and $\Box\Diamond C^\epsilon$ are the specifications for safety, reachability and persistence, respectively, with respect to constraints $C$ and error $\epsilon$. Note that it is important to specify a desired property with some error, since many constraint methods cannot "solve" the constraint satisfaction problem in finite time (or with finite resource). A more formal notion of this kind of *open specification* is introduced in [12] and discussed in [17].

The strongest constraint-based requirement is the safety property which requires that the system never move away from the solution set of the constraints. The weakest constraint-based requirement is the persistence property which allows the system to be only asymptotically stable at the solution set of the constraints. Between these two extremes, there is the reachability property which ensures that the system approaches the solution set of the constraints eventually.

For example, the state variables of the robot soccer player (Fig. 3) are $(x, y, \theta)$. Let the desired goal be $(x_d, y_d, \theta_d)$. The set of constraints will be $x = x_d, y = y_d, \theta = \theta_d$. Let $G$ denote an $\epsilon$-neighborhood of $(x_d, y_d, \theta_d)$. A constraint-based specification for the soccer player is the persistence property $\Box\Diamond G$.

As another example, if the boundary of the soccer field is described by a manifold in the configuration space [9] of the car $f(x, y, \theta) = 0$, a safety property for the soccer player could be $\Box f(x, y, \theta) > 0$, that is, the car is always on the field. The corresponding constraint for this specification is the inequality $f(x, y, \theta) \geq \epsilon$ for some $\epsilon > 0$.

## 5 Constraint-Based Control

*How does the controller satisfy the constraints?*

Given a constraint-based requirements specification, the design of the controller becomes the synthesis of an embedded constraint solver which, together with the dynamics of the plant and the environment, solves the given constraints on-line.

### 5.1 Constraint methods

We have viewed constraint satisfaction as a dynamic process that approaches the solution set of the constraints [20]. We have also specified this constraint-based behavior using TLTL and timed $\forall$-automata [18]. Let $C$ be a set of constraints, $sol(C)$ be the solution set of $C$ and $C^\epsilon$ be the $\epsilon$-neighborhood of $sol(C)$; a constraint net $CN$ is a *constraint solver* for $C$, iff $\forall \epsilon > 0$, $[\![CN]\!] \models \Diamond\Box C^\epsilon$. For example, a constraint net corresponding to $\dot{x} = -x$ is a constraint solver for $x = 0$.

There are typically two kinds of constraint satisfaction problems, namely, global consistency and optimization [20]. The problem of *global consistency* is to find a solution tuple that satisfies all the given constraints. The problem of *unconstrained optimization* is to minimize a function $\mathcal{E} : \mathcal{R}^n \rightarrow \mathcal{R}$. Global consistency is for solving *hard* constraints and unconstrained optimization is for solving *soft* constraints. A problem of the first kind can be translated into one of the second by introducing an energy function representing the degree of global consistency. For example, given a set of equations $g(x) = 0$, let $\mathcal{E}_g(x) = \| g(x) \|_2$ where $\| \cdot \|_2$ is a quadratic norm. If a constraint solver $CN$ solves $\min \mathcal{E}_g(x)$, $CN$ solves $g(x) = 0$ if a solution exists.

Various constraint methods fit into our framework for constraint satisfaction [20]. There are two classes of constraint methods, *discrete relaxation* which can be implemented as state transition systems, and *differential optimization* which can be implemented as state integration systems. Both can be modeled as constraint nets [20]. We will illustrate these methods with two examples. For more comprehensive analysis of these methods, the reader is referred to [20].

*Newton's method* [16] is a typical discrete relaxation method that minimizes a second-order approximation of the given function at each iterative step. Let $\bigtriangledown\mathcal{E} = \frac{\partial \mathcal{E}}{\partial x}$ and $J$ be the Jacobian of $\bigtriangledown\mathcal{E}$. At each step with current point $x^{(k)}$, Newton's method minimizes the function:

$$\mathcal{E}_a(x) = \mathcal{E}(x^{(k)}) + \bigtriangledown\mathcal{E}^T(x^{(k)})(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T J(x^{(k)})(x - x^{(k)}).$$

Let $\frac{\partial \mathcal{E}_a}{\partial x} = 0$, we have:

$$\bigtriangledown \mathcal{E}(x^{(k)}) + J(x^{(k)})(x - x^{(k)}) = 0.$$

The solution of the above equation becomes the next point, i.e.,

$$x^{(k+1)} = x^{(k)} - J^{-1}(x^{(k)}) \bigtriangledown \mathcal{E}(x^{(k)}).$$

Newton's method defines a state transition system, with the next state as a function of the current state.

The *gradient method* [14] is a typical differential optimization method, based on the gradient descent algorithm, where state variables slide downhill in the direction opposed to the gradient. Formally, if the function to be minimized is $\mathcal{E}$, the vector that points in the direction of maximum increase of $\mathcal{E}$ is the gradient of $\mathcal{E}$. Therefore, let $\bigtriangledown \mathcal{E} = \frac{\partial \mathcal{E}}{\partial x}$, the following gradient descent equation models the gradient method

$$\dot{x} = -k \bigtriangledown \mathcal{E}(x), \quad k > 0. \tag{5}$$

Various constraint methods [13, 20] can be applied to control synthesis under this framework. More importantly, as we have shown in [20], most constraint methods are associated with an energy function, which can be directly used as a Liapunov function in behavior verification [18].

## 5.2 Constraint-based controllers

Constraint solvers are closed nets, with all state variables controllable (outputs). Controllers are open systems, with both sensory inputs and controllable outputs. We call a controller an *embedded constraint solver* if the controller, together with the plant and the environment, satisfies the given constraint-based specification.

Most constraint methods can be applied to embedded constraint solvers as well. Embedded constraint solvers can be either discrete or continuous according to the constraint method used. Continuous solvers, based on the gradient method, generalize potential functions [6]. Discrete solvers, based on numerical or symbolic relaxation methods, are more flexible in many applications.

If Newton's method or the gradient method is used, the design of an energy function depends on the type of constraint-based specification. For reachability or persistence constraints, the energy function defines the degree of satisfaction of the constraints; for safety constraints, the energy function defines the degree of satisfaction of the constraints within $C^\epsilon$ while the energy outside of $C^\epsilon$ becomes infinity. For example, given a persistence specification $\Box\Diamond C^\epsilon$ with $C$ defined as $f(x) = 0$, an energy function for this specification can be $\frac{1}{2}f^2(x)$. If $\Box(f(x) > 0)$ is required, an energy function can be $\max(-\ln \frac{f(x)}{\epsilon}, 0)$, i.e., if $f(x) \geq \epsilon$, then $\mathcal{E}(x) = 0$, if $0 < f(x) < \epsilon$, then $\mathcal{E} > 0$, and if $f(x) \to 0$, then $\mathcal{E}(x) \to \infty$.

Various existing controllers, from simple linear proportional derivative (PD) control to complex nonlinear adaptive control or potential field methods [6], can be derived and analyzed in this framework. Various learning algorithms and neural net computations [14] can be formalized in this framework as well.

In order to synthesize a controller for the robot soccer player, we must define a mapping from the plant state variables $\langle x, y, \theta \rangle$ to the control variables $\langle v, \alpha \rangle$, so that the system satisfies $\Box\Diamond C^\epsilon$ where $C$ is the set of constraints $\{x = x_d, y = y_d, \theta = \theta_d\}$. We use the gradient method and define an energy function for the controller as

$$\mathcal{E} = \frac{k_p}{2}(x_d - x)^2 + \frac{k_p}{2}(y_d - y)^2 + \frac{k_t}{2}(\theta_d - \theta)^2.$$

The controller is designed to make $\mathcal{E}$ go to its minimum.

The derivation of the controller is as follows. Let $p$ be the length of the path. We have $v = \dot{p}$. Using the gradient method on the energy function $\mathcal{E}$, we have

$$\dot{p} = -k_1 \frac{\partial \mathcal{E}}{\partial p}, \;\; \dot{\theta} = -k_2 \frac{\partial \mathcal{E}}{\partial \theta}$$

where $\frac{\partial \mathcal{E}}{\partial p}$ and $\frac{\partial \mathcal{E}}{\partial \theta}$ can be computed as follows:

$$-\frac{\partial \mathcal{E}}{\partial p} = k_p(x_d - x)\frac{\partial x}{\partial p} + k_p(y_d - y)\frac{\partial y}{\partial p} + k_t(\theta_d - \theta)\frac{\partial \theta}{\partial p}$$

where

$$\frac{\partial x}{\partial p} = \frac{\dot{x}}{v} = \cos(\theta), \;\; \frac{\partial y}{\partial p} = \frac{\dot{y}}{v} = \sin(\theta), \;\; \frac{\partial \theta}{\partial p} = \frac{\dot{\theta}}{v} = \frac{\tan(\alpha)}{L}$$

and

$$-\frac{\partial \mathcal{E}}{\partial \theta} = k_p(x_d - x)\frac{\partial x}{\partial \theta} + k_p(y_d - y)\frac{\partial y}{\partial \theta} + k_t(\theta_d - \theta)$$

where

$$\frac{\partial x}{\partial \theta} \doteq -v\sin(\theta), \;\; \frac{\partial y}{\partial \theta} \doteq v\cos(\theta).$$

Let $d = \sqrt{(x_d - x)^2 + (y_d - y)^2}$ and $\theta' = \tan^{-1}(y_d - y, x_d - x)$. According to the dynamics of the plant modeled by Equation (1), we have $\alpha = \tan^{-1}(\frac{L}{v}\dot{\theta})$. Therefore, the control law for the tracking problem is:

$$\begin{aligned} v &= k_1[k_p d\cos(\theta' - \theta) + k_t(\theta_d - \theta)\frac{\tan(\alpha)}{L}] \\ \alpha &= \tan^{-1}(\frac{L}{v}k_2(k_p v d\sin(\theta' - \theta) + k_t(\theta_d - \theta))). \end{aligned}$$

Now we are able to analyze this control law using the energy function for the controller as the Liapunov function of the system. Since

$$\begin{aligned} \dot{\mathcal{E}} &= -[k_p(x_d - x)\dot{x} + k_p(y_d - y)\dot{y} + k_t(\theta_d - \theta)\dot{\theta}] \\ &= -[k_p(x_d - x)\cos(\theta) + k_p(y_d - y)\sin(\theta) + k_t(\theta_d - \theta)\frac{\tan(\alpha)}{L}]v \\ &= -\frac{1}{k_1}v^2 \le 0, \end{aligned}$$

the Liapunov function is nonincreasing, therefore, the system is "stable." However, strictly speaking, the controller is not an embedded constraint solver since the system is not asymptotically stable at the solution: if $|\theta' - \theta| = \frac{\pi}{2}k$ and $\theta_d = \theta$ we get $v = 0$ even when $d \neq 0$. We prove that they are the only singularities of this control law.

**Proposition 5.1** *This control law satisfies the condition $v = 0$ iff*

$$(d = 0 \vee |\theta' - \theta| = \frac{\pi}{2}k) \bigwedge (\theta_d = \theta).$$

Proof: According to the control law for $\alpha$, $v = 0$ implies $\theta_d = \theta$. According to the control law for $v$, $v = 0$ implies $d\cos(\theta' - \theta) = 0$. □

Therefore, the car can stop at singularities without reaching the desired goal. In order to solve this problem, we use a hierarchical control structure.

## 6 Robotic Architecture

*How should we compose a control system?*

A control system, in general, is implemented in a vertical hierarchy [1] corresponding to a hierarchical abstraction of time and domains (Fig. 5). The bottom level sends

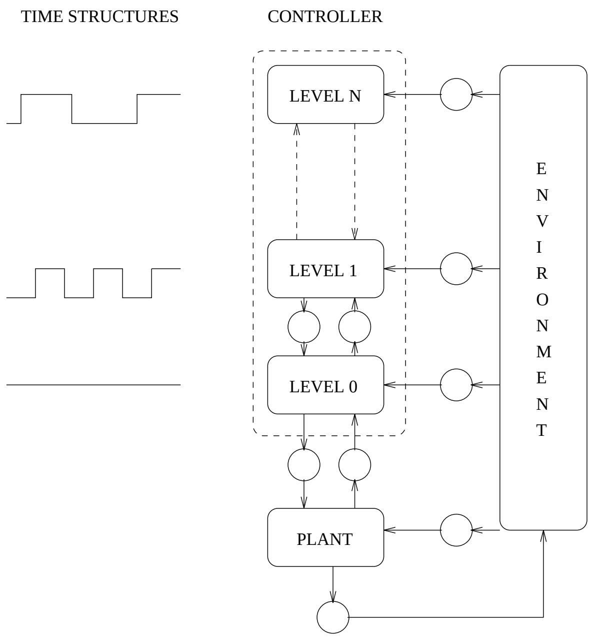

**Fig. 5.** Abstraction hierarchy

control signals to various actuators, and at the same time, senses the state of actuators. Control signals flow down and the sensing signals flow up. Sensing signals from the environment are distributed over levels. Each level is a black box that represents the causal relationship between the inputs and the outputs. The inputs consist of the control signals from the higher level, the sensing signals from the environment and the current states from the lower level. The outputs consist of the control signals to the lower level and the current states to the higher level. Usually, the bottom level is implemented by analog circuits that function with continuous dynamics and the higher levels are realized by distributed computing networks.

In our framework of control synthesis, constraints are specified at different levels on different domains, with the higher levels more abstract and the lower levels more plant-dependent.

A control system can be synthesized as a hierarchy of interactive embedded constraint solvers. Each abstraction level solves constraints on its state space and produces the input to the lower level. The higher levels are composed of digital/symbolic event-driven control derived from discrete constraint methods and the lower levels embody analog control based on continuous constraint methods.

For the robot soccer player, we have designed a two level controller. The low level tracks the given goal position $(x_d, y_d, \theta_d)$ in continuous time and the high level, driven by events from below, sets a new goal position according to the current ball position, the target position and the boundary of the field. Obstacle and singularity avoidance are considered at this level too. We will not describe this level in any detail here. There are various kinds of events to trigger the high level planner; the current version uses a simple event: the car coming to a stop (which implies that either the current goal is achieved or the car is stuck because of singularity or obstacles). Other events such as the ball moving significantly or an obstacle obstructing the path can be incorporated. Fig. 6 shows an overall CN control structure, with the models of the car (plant) and the ball (environment).

## 7 Developing Hybrid Control Systems

*Where are we and where are we going?*

We have considered control synthesis as one of the four key problems in developing hybrid control systems, together with modeling, specification and verification.

We have developed a unified framework for synthesizing continuous/discrete hybrid control systems based on a simple principle: on-line constraint satisfaction. However, we are not aiming to replace existing control theory, rather to formalize and generalize the underlying principles that are used in practice.

Local minima and/or singularities can be a major problem for this type of controller. In general, a complex robot control system should be developed and organized hierarchically. Normally singularities can be avoided if a higher level control strategy is used to detect singularities and to produce a sequence of intermediate configurations between the actual and the target configurations. Such a higher level event-triggered control strategy becomes more important when the robot is embedded in a complex environment.

We have developed a modeling and simulation environment, called ALERT (A Laboratory for Embedded Real-Time systems), in which the robot soccer player has been modeled and simulated. We are experimenting with an implementation on the real soccer players.

In the future, we also plan to characterize some restricted classes of hybrid control systems and investigate semi-automatic or automatic control synthesis methods for

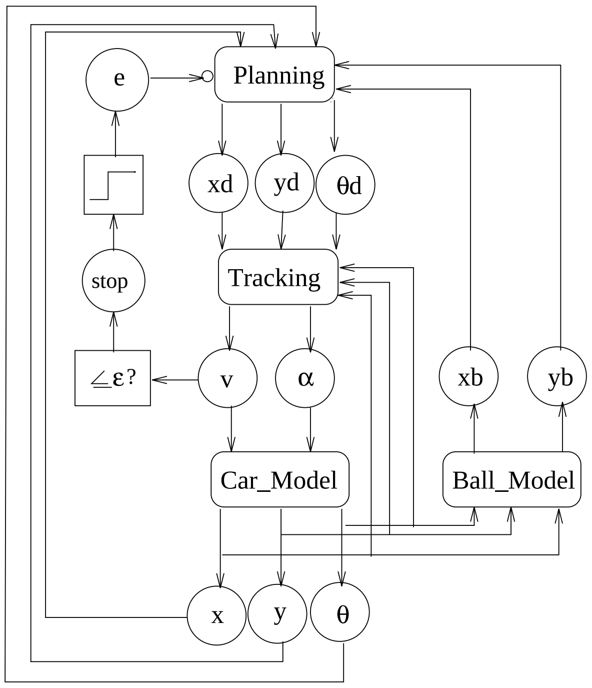

**Fig. 6.** A control structure for the robot soccer player

these classes. The fundamental goal is to provide foundations for the design and analysis of robotic systems and behaviors.

## References

1. J. S. Albus. *Brains, Behavior, and Robotics*. BYTE Publications, 1981.
2. E. A. Ashcroft. Dataflow and eduction: Data-driven and demand-driven distributed computation. In J. W. deBakker, W.P. deRoever, and G. Rozenberg, editors, *Current Trends in Concurrency*, number 224 in Lecture Notes on Computer Science, pages 1 – 50. Springer-Verlag, 1986.
3. A. Benveniste and P. LeGuernic. Hybrid dynamical systems theory and the SIGNAL language. *IEEE Transactions on Automatic Control*, 35(5):535 – 546, May 1990.
4. P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *ACM Proceedings on Principles of Programming Languages*, pages 178 – 188, 1987.
5. E. Emerson. Temporal and modal logic. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, MIT Press, 1990.
6. D. E. Koditschek. Robot planning and control via potential functions. In J. Craig O. Khatib and T. Lozano-Perez, editors, *The Robotic Review 1*. MIT Press, 1989.
7. J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

8. J. Lavignon and Y. Shoham. Temporal automata. Technical Report STAN-CS-90-1325, Robotics Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305, 1990.
9. T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2), February 1983.
10. Z. Manna and A. Pnueli. Specification and verification of concurrent programs by $\forall$-automata. In *Proc. 14th Ann. ACM Symp. on Principles of Programming Languages*, pages 1–12, 1987.
11. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.
12. A. Nerode and W. Kohn. Models for hybrid systems: Automata, topologies, controllability, observability. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, number 736 in Lecture Notes on Computer Science, pages 317 – 356. Springer-Verlag, 1993.
13. D. K. Pai. Least constraint: A framework for the control of complex mechanical systems. In *Proceedings of American Control Conference*, pages 426 – 432, Boston, 1991.
14. J. Platt. Constraint methods for neural networks and computer graphics. Technical Report Caltech-CS-TR-89-07, Department of Computer Science, California Institute of Technology, 1989.
15. M. Sahota and A. K. Mackworth. Can situated robots play soccer? In *1994 Canadian Artificial Intelligence, Banff, Alberta*, May 1994.
16. J. T. Sandfur. *Discrete Dynamical Systems: Theory and Applications.* Clarendon Press, 1990.
17. Y. Zhang. A foundation for the design and analysis of robotic systems and behaviors. Technical Report 94-26, Department of Computer Science, University of British Columbia, 1994. Ph.D. thesis.
18. Y. Zhang and A. K. Mackworth. Specification and verification of constraint-based dynamic systems. In A. Borning, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science 874, pages 229 – 242. Springer Verlag, 1994.
19. Y. Zhang and A. K. Mackworth. Constraint Nets: A semantic model for hybrid dynamic systems. *Theoretical Computer Science*, (138):211 – 239, 1995. Special Issue on Hybrid Systems.
20. Y. Zhang and A. K. Mackworth. Constraint programming in constraint nets. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*, pages 49 – 68. MIT Press, 1995.