

On-line, Reflexive Constraint Satisfaction for Hybrid Systems: First Steps

(Invited Presentation)

Michael S. Branicky

Dept. of Electrical Engineering and Applied Physics, Case Western Reserve
University, 10900 Euclid Ave., Glennan 515B, Cleveland, OH 44106-7221 USA.
E-mail: branicky@eeap.cwru.edu.

Abstract. We can achieve guaranteed constraint satisfaction of a hybrid dynamical system (which takes into account the underlying continuous dynamics) in a simple, hierarchical control algorithm. Two layers of functionality, (1) piece-wise viable servo controllers and (2) a “reflex controller,” are required for guaranteed constraint satisfaction. The resulting control structure allows higher-levels of “intelligence” or functionality to be added which don’t have to worry about guaranteeing constraints. The structure acts as an on-line filter which approves any actions that will maintain constraints but denies any requests that would result in behavior outside that specified.

In this note, we lay down the notation and theory of this hierarchy in a broad, abstract setting. Minimal properties to assure constraint satisfaction are given. The incorporation of such a model with higher planning levels and its associated convergence properties are discussed. The philosophy has been successfully applied to a robot’s maintaining collision avoidance while under higher-level control, viz. path planners and teleoperation. The system operates on-line in real-time; in executing collision-free motions, the robot uses its full mechanical bandwidth.

1 Introduction

Intelligent, autonomous systems require knowledge about themselves and their environment and the ability to integrate this knowledge to make decisions about how to act. Ultimately, these decisions may be as complex as those we make on a daily basis. But such abilities were long evolved and built up from robust underlying layers of functionality.

If we are ever to achieve such complex behaviors in systems, then we must build the robust intervening layers first: those which connect higher-level planning and lower-level servo control. One possible level of intelligence that might sit between servo control and higher layers of intelligence is called *reflex control*. Reflex control (or reflexes) obtains its name from the analogy to animal reflexes—and its functionality is almost identical. Reflexes are always there to protect us, though most of the time their influence is minimal.

Consider the act of picking up a pot on the stove. You plan to pick the pot off the burner and place it on a plate. On a lower level, your brain commands your muscles to perform the task. If all is right, you perform the task as planned. However, if the pot is too hot, your reflexes will override the “higher level planning” and command your hand to move quickly away from the danger of getting burned. A low level of intelligence (your reflexes) refuses to carry out the “higher level” plans, saving you some pain. However, they only act when it is necessary to do so. Your hand does not jerk away from a cold pot. They are transparent if no danger is present. They are minimally influential.

Two important points have been brought forward: the reflexes should be transparent unless needed; they should keep the system from violating constraints. The ramifications of these two statements are far-reaching. The reflexes must be very fast to be transparent to most control schemes. They must be able to make decisions about safe or unsafe moves. They must be aware of the system and its environment.

In this paper, we begin to deal with the problem of assuring operation of a hybrid dynamical system within specifications, taking into account the underlying continuous dynamics. The result is a hierarchical control structure which guarantees specification satisfaction with a “reflex controller,” under mild conditions imposed on the plant’s lower-level “servo” controllers. Essentially, the reflex controller acts like a filter of higher-level commands, passing through requests that maintain constraints and denying any that would result in behavior outside of specifications. Thus, the analogy with animal reflexes: a control layer which is always present but only initiates action when it is necessary to do so for self-preservation. Although animal reflexes are inherently parallel, we use a sequential filter implementation to mimic parallel operation because it guarantees safety even if the reflex controller fails (or takes an inordinate amount of time to make a decision).

For brevity, we refer to behavior outside of specifications as a *specification violation* or, simply, a *violation*. We call sets of violation points *obstacles*. Thus, the reflex controller only acts (denies requests) when it is necessary to do so to prevent violation (imminent because of known constraints or possible because an area has yet to be approved as satisfying specifications).

Setting. Herein, we consider as “systems” *hybrid dynamical systems* as follows [5]:

$$\begin{aligned} \dot{x}(t) &= f(x(t), q(t), u(t)), & \text{if } (x, q) \in S \setminus A \\ [x(t^+), q(t^+)] &= G(x(t), q(t)), & \text{if } (x, q) \in A \end{aligned} \quad (1)$$

where $x(t) \in X \subset \mathbf{R}^n$, $q \in Q \simeq \{1, 2, \dots, N\}$, and $u(t) \in U \subset \mathbf{R}^m$. The *hybrid state* of this system is $s = (x, q)$, taking values in $S = X \times Q$, with $A \subset S$ closed. We may also associate the *output equation* $y = h(s, u)$ taking values in $Y \subset \mathbf{R}^p \times \mathbf{Z}^r$. Thus, starting at $[x_0, i]$, the continuous state trajectory $x(\cdot)$ evolves according to $\dot{x} = f(x, i, u)$. If $(x(\cdot), i)$ hits A at time t_1 , then the state becomes $G(x(t_1), i) = [x(t_1^+), j]$, from which the process continues.

Constraint space (C-space) is the multi-dimensional space of generalized co-

ordinates which represent the variables of interest of a system.¹ For example, a variable of interest may be an output that must be maintained within certain (possibly time-varying) bounds. We call a point in this C-space a *configuration*.

For the moment, let us consider the C-space of the plant and its associated controllers at time t to be the set, $\mathcal{C}(t)$. Then, to use C-space for constraint satisfaction, one associates a mapping from C-space to the set $\{0, 1\}$ as follows

$$O : \mathcal{C}(t) \mapsto \{0, 1\}$$

$$O(c(t)) = \begin{cases} 0, & \text{if the system in configuration } c(t) \text{ satisfies constraints,} \\ 1, & \text{if the system in configuration } c(t) \text{ violates constraints.} \end{cases} \quad (2)$$

The Problem. With this map, constraint satisfaction is reduced to point navigation in the set $(O(t))^{-1}(0)$. In particular, if we let $c(t)$ denote the configuration of our system at time t and *free-space* at time t , $F(t)$, denote $O^{-1}(0)$ at time t , then we are interested in

Problem 1 (Constraint-Satisfaction or Obstacle-Avoidance). Control the system so that there are no violations: maintain

$$c(t) \in F(t), \quad \text{for all } t \geq t_0.$$

The rest of this paper deals with Problem 1. Below, we will refer to any set $S \subset O^{-1}(0)$ as being *violation-free* or *obstacle-free*.

Example 1 (Anti-windup). Consider the system

$$\dot{e} = \xi,$$

where ξ is some error signal we are accumulating through integration. In many situations, one prevents “integrator windup” by limiting the value of e between some bounds L and H : $F(t) \equiv F = \{e \mid L \leq e \leq H\}$.

Example 2 (Ball in a Box). Consider the system

$$\ddot{x}(t) = u; \quad v(t^+) = -v(t), \quad \text{if } x \in \{-1, 1\}$$

defined on $X = [-1, 1]$, where $v \equiv \dot{x}$, and $U = [-a_{\text{sat}}, a_{\text{sat}}]$. To specify no (non-zero velocity) collisions, set $O(t) \equiv O = \{-1, 1\}$.

For ease of presentation, we will only deal here with the case of static constraints as in the examples above, i.e., $F(t) \equiv F$ for all t , even though extensions to the time-varying case are possible.²

With this “constraint,” it may appear that Problem 1 is a null problem—at least in Examples 1 and 2 (just set $\xi = u = 0$). Hence, completely analogous to safety and liveness in computer science [17], we refine to

¹ Those familiar with collision avoidance in robotics will notice the connections with configuration space [11, 12]. See below.

² For instance, to assure constraint specification, it is only necessary that the C-space “obstacles” arising from new or moving constraints remains outside the current valid bounding setpoint set of the reflex controller (terms defined below).

Problem 2 (Reflexive Constraint Satisfaction or Reflexive Obstacle Avoidance). Control the system so that we have as much freedom to move in $F(t)$ as possible: any point in $F(t)$ that can be visited without irrecoverably causing violation should be attainable.

Hence, in Problems 1 and 2, we have somewhat formalized the dual nature of reflex control mentioned above: keep the system safe, but deny action only when it is absolutely necessary to do so.

Example 3 (Reflexive Anti-windup). Allow

$$\xi = \begin{cases} 0, & \text{if } \xi_{\text{request}} > 0 (\text{resp. } < 0) \text{ and } e = H (\text{resp. } L), \\ \xi_{\text{request}}, & \text{otherwise.} \end{cases}$$

Example 4 (Reflexive Boxed Ball). Let $b(t) = v^2(t)/(2a_{\text{sat}})$ and $d(t) = \text{sgn}(v(t))$. Allow

$$u = \begin{cases} -d(t)a_{\text{sat}}, & \text{if } |x(t) + d(t)b(t)| = 1, \\ u_{\text{request}}, & \text{otherwise.} \end{cases}$$

The above examples are illustrative, but quite simple. It is not hard to think of cases of a ball moving in multiple dimensions, among multi-dimensional obstacles, in which the global goal of the system can be more closely achieved by switching among different approved subsets of the C-space (cf. Figure 1). This will occur in general: different local domains of attraction will be associated with different controllers; only by stringing them together will more global goals be accomplished with safety. In general, switching may also arise due to computational considerations such as inspection time of C-space regions for obstacles or conservatism in estimating dynamic bounds.

The reader interested in more complicated examples may wait for our description of robotic collision avoidance (simplified from [3, 16]) or consider an example arising in flight control [4].

Organization. In the next section, we present a motivating example: robotic collision avoidance. In Section 3, we detail generic conditions and “algorithms” for on-line, reflexive constraint satisfaction in a hierarchical framework. In Section 4 we analyze the case of robotic collision avoidance in our general framework, providing an example of the theory we have found useful in practice. Conclusions are in Section 5.

2 Motivation: Robotic Collision Avoidance

Real-time obstacle avoidance for high-speed robots has been achieved via the *reflex control* concept [3, 13, 14, 15, 16]. The reflex layer stands between servo control and planning in the “intelligence hierarchy” of the robot. It will transparently execute the commands given by the higher level unless it is dangerous to do so. It will override commands that would result in the manipulator’s colliding with itself or objects in its environment.

The reflex control layer can be introduced in either a serial or parallel manner. In a parallel implementation, the reflexes would monitor the state of the robot (positions and velocities) and the state of the world (represented in configuration space) and “step in” whenever it is necessary to specify commands that will ensure continued safe operation. That is, the reflexes are always present but intervene only when it is necessary to prevent imminent danger. It is this type of operation that prompts the analogy with biological reflexes.

The reflex control can be implemented in series with the same result: it can receive requests from higher layers and either deny or approve them based on whether a collision is imminent or not. This is the implementation adopted.

Conceptually, reflex control is consistent with layered, hierarchical control systems [2, 6]. In a hierarchical control structure, the reflex control layer is inserted between the layers of motor control and higher layers, such as path planning. They are merely a limited form of intelligence which prevents the robot from exceeding joint limits and from striking itself or objects in its environment. They are low in the control hierarchy, but that does not limit their effects. Just as motor saturation might prevent a linear control law from producing the torques it requests, so do the reflexes intervene to deny any requests from higher levels which would result in collision. And just as the effect of saturation is transparent to torque requests below the saturation level, so too should the reflexes be transparent when requested paths are not dangerous.

2.1 Analysis Preliminaries

Configuration Space. As noted before, configuration space (*C-space*) is a multi-dimensional space of generalized coordinates which represent the *degrees of freedom* (*d.o.f.*) of a system. The configuration space used in robotics is the joint space of the robot; the generalized coordinates used are the robot’s joint parameters [11, 12].

For the moment, let us consider the C-space of the robot to be the set, \mathcal{C} , of all poses or configurations of our robot. Then, to use C-space for obstacle avoidance, one associates a mapping from C-space to the set $\{0, 1\}$ as in Equation (2), with the constraint being “the robot intersects no obstacles.” Thus, obstacle avoidance is reduced to point navigation in the set $O^{-1}(0)$.

For example, consider a robotic manipulator having n links with q_i ($i = 1, \dots, n$) representing the generalized coordinate of the i th joint. Since each joint parameter represents a degree of freedom, the configuration space of the manipulator is an n -dimensional space with orthogonal axes measuring displacements of the q_i ’s. Each point $q = (q_1, \dots, q_n)$ in the configuration space corresponds to a unique pose or “configuration” of the manipulator. If each $q_i \in [\theta_{i_{\min}}, \theta_{i_{\max}}]$, then the configuration space, \mathcal{C} , of the manipulator is given by the following Cartesian product [7]:

$$\mathcal{C} = [\theta_{1_{\min}}, \theta_{1_{\max}}] \times [\theta_{2_{\min}}, \theta_{2_{\max}}] \times \dots \times [\theta_{n_{\min}}, \theta_{n_{\max}}] \subset \mathbf{R}^n.$$

If the robot in question has no joint limits (or has rotational limits further than

2π apart), then one may want to associate points modulo full rotation and use S^1 instead of a closed interval in that coordinate.

Prism Notation. In a slight abuse of common notation (including that used above), we will use $[a, b]$ to denote the set of real numbers “between” a and b . If $a, b \in \mathbf{R}^1$, then

$$[a, b] = \{x \mid a \leq x \leq b, \text{ if } a \leq b; b \leq x \leq a, \text{ otherwise}\}.$$

If $a, b \in S^1$, then $[a, b] = \{x \mid \text{one of } a \leq x \leq b; b \leq x \leq a\}$, where the desired interval has been decided upon by some higher level, e.g., preference of direction or a global planner. Finally, if $a, b \in \mathcal{C}$, then

$$[a, b] = \times_{i=1}^n [a_i, b_i],$$

that is, the hyperprism generated by a and b . For convenience, we often use “prism” instead of “hyperprism.”

Dynamics and Control. Let us assume dynamic decoupling of our robot, either through feedback or design. In this case, the dynamics of each subsystem may be described as

$$\dot{x} = v, \quad \dot{v} = u, \quad u \in [-a_{\text{sat}}, a_{\text{sat}}]. \quad (3)$$

To be specific, we assume a proportional plus derivative (PD) controller as follows:

$$u = K_p(x_{\text{setpt}} - x) - K_v(v_{\text{setpt}} - v), \quad (4)$$

where x_{setpt} is the desired position and K_p and K_v are chosen for critical- or over-damping. Hence, starting from any $(x_0, 0)$ with $v_{\text{setpt}} \equiv 0$, the robot will travel to x_{setpt} without overshoot, i.e., without passing to the “side” of x_{setpt} opposite from x_0 . Starting from $(x(t), v(t))$, $v(t) \neq 0$, we denote the “closest” point x_{setpt} which results in no overshoot as the *braking point*, $b(t)$; we denote by $a(t)$ the “closest” point x_{setpt} which results in maximum acceleration (actuator saturation).

2.2 Implementing Reflex Control

Point-to-Point Mode. In point-to-point operation, the reflexes accept position requests from higher layers and output position setpoints to a servo controller. Assume, for well-posedness, that no obstacles are in the prism $[x(t_0), b(t_0)]$. Assume also PD control as in Equation (4) and with $v_{\text{setpt}} \equiv 0$, and with K_d and K_v chosen for critical damping, i.e., $K_p = \omega_n^2$, $K_v = 2\omega_n$. Taking into account velocity and actuator saturation, the maximum acceleration setpoint, $a(t)$, and the braking point, $b(t)$, may be explicitly computed. See [15] for details.

The bottom line is, given a request position, x_{request} , we must search the prism $[x(t), x_{\text{request}}]$ for obstacles if we are to approve x_{request} as a setpoint. But searching may take time. It makes no sense, however, to search farther than $a(t)$. Hence, we search may search $[x(t), a(t)]$, set $x_{\text{setpt}}(t_i) = a(t)$, and then search $[a(t_i), a(t)]$ on the next update. If we ever find an obstacle present,

then the setpoint we approve is the free one “closest” to x_{request} . Since this point was outside of $b(t)$ by construction, the robot will be able to stop without overshoot—hence without hitting the obstacle—at this point.

Direct Acceleration Control Emulation. Reflex control using direct acceleration control emulation assumes the existence of some acceleration-based controller at a higher level. It assumes a servo controller that accepts position and velocity commands from the reflex module, and issues an acceleration command in terms of an anticipatory setpoint to a simple PD controller as in Equation (4).

The reflex controller invents a virtual setpoint which has the result of converting acceleration requests into acceleration commands to the linearized, decoupled plant. Specifically, acceleration commands will track acceleration requests if the reflex controller chooses

$$x_{\text{setpt}} = x + \frac{K_v}{K_p}v + \frac{u_{\text{request}}}{K_p}; \quad v_{\text{setpt}} = 0.$$

This transformation of acceleration commands into equivalent instantaneous position commands permits a geometric specification of the reflex algorithm. That is, an acceleration request is granted by approving the corresponding setpoint, and this setpoint is approved if it would not result in a collision under PD control (as described above). See [16] for details.

Reflex Control in Higher Dimensions. Our description of reflex control to this point has been limited to one dimension. In only one dimension, the search for potential collisions in discretized configuration-space consists of examining all C-Space elements in line between the current position and the goal position.

Reflex control in one dimension is generalizable to higher dimensions. If a decoupled plant is assumed, then each second order system may be treated like the one-dimensional system described above. The difference with higher dimensions is that the regions which must be inspected for obstacle avoidance are no longer simple distances, but areas, volumes, or hypervolumes whose vertices are given by $x_i(t)$ to $b_i(t)$ (or $a_i(t)$) in each dimension $i = 1, \dots, n$.

Another difference with higher dimensions is the effect of *geometric coupling* among the subsystems [14, 15]. This effect is shown in Figure 1. In order to avoid choosing the overly conservative setpoint, some choice must be made as to which axis gets precedence. This choice may be made based on distance to the goal point or some other optimizing criterion. In previous implementations, however, a simple list of joint precedences was used to eliminate such conflicts.

The same figure, assuming a goal point to the upper right of the obstacle, demonstrates the need to switch among different “local” setpoints in order to accomplish global tasks.

3 Generic Conditions for Reflexive Obstacle Avoidance

In this section we detail generic properties or conditions used in the construction and analysis of our hierarchical controller.

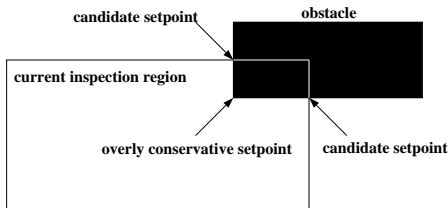


Fig. 1. Geometric Coupling in Obstacle Avoidance

We wish to build a generic hierarchical structure which provides guaranteed constraint satisfaction. For the moment, we separate the control structure of our system into three abstract levels:

1. Lower-level “servo” controller
2. Reflex controller
3. Higher-level processes

The reflex controller can be further broken down into (1) an active reflex controller and (2) a C-space inspector or approver.

These levels interact by passing objects called *bounding sets* (or, simply, *sets*). In particular, a higher-level process may, from time to time, pass a set, X , to the reflex controller.³ We call these passings *requests*. Likewise, the reflex controller passes sets to the servo controller. We call these passings *setpoints*. Internal to the reflex controller, the C-space inspector passes the active reflex sets which it has certified as violation-free called *approvals*. Below, $S(t)$ and $R(t)$ represent the setpoint and request sets at time t , respectively.

3.1 Preliminaries

We first need a few definitions.

Definition 3 (Control Policy). A *control policy* is simply an achievable prescription for controlling the system from its current state: $u = F(s)$.

This is a general concept. The means to the end is completely arbitrary. The important thing is that the policy be physically achievable.

Example 5 (Braking Policies). In controlling a robot, we are not allowed to “stop on a dime.” Typically, one has in mind only a single control policy for braking, such as “full braking in all joints,” but it may be advantageous to consider several different braking policies. Another would be to use Equation (4); note we obtain a different policy for each value of K_d , K_v , and x_{request} .

³ In a computer implementation a finite description of the set is passed. For instance, the prism coordinates x and y instead of the actual set $[x, y]$.

Theoretically, one could have an infinite number of control policies and we allow that in our abstract formulation: we talk about the generic control policy $\alpha \in P$, where P is the set of all implementable⁴ control policies. Ultimately, though, we have to invoke particular control policies in response to given situations. Further, these invocations must be done in a timely manner if we are to guarantee obstacle avoidance while allowing behavior near the performance limitations of the system. Thus, there will typically be a tradeoff between our ability to deal with the possibilities of different control policies and the conservatism of the system’s allowed dynamical behavior.

Control policies are particularly useful if the state space trajectories arising from their invocation are known, computable, or able to be bounded. In any case, however, we define the *sweep set* as follows:

Definition 4 (Sweep Set). The *sweep set corresponding to control policy α from s* is the set

$$B_\alpha(s) = \{y(s(\tau)), \tau \geq t, \text{control policy } \alpha \text{ is invoked at time } t \text{ from state } s(t)\}$$

that is, the curve in C-space resulting from initiating control policy α at s .

Just as we are interested in sets that are violation-free, we are also interested in sets which satisfy a particular dynamic property important in collision avoidance:

Definition 5 (Valid Set). A set X containing $B_\alpha(s)$, $\alpha \in P$, is said to be *valid for control policy α at s* or simply *α -valid at s* . We say that set X is *valid at s* if there exists an $\alpha \in P$ such that X contains $B_\alpha(s)$.

Note: We drop the phrase “at s ” when s is clear from context.

Example 6. Consider the planar system

$$(\dot{x}, \dot{y}) = (1 - \alpha, \alpha),$$

with $\alpha(t) \in \{0, 1\}$. Then, lines parallel to the x -axis are 0-valid; lines parallel to the y -axis are 1-valid. The nonnegative x -axis is 0-valid at $(0, 0)$.

Thus, a set is valid if there exists an implementable control policy such that, if it were invoked at time t , the system remains in that set. To get a feeling for the concept, we make several easily proven remarks:

- Remark.*
1. If A is valid and $B \supset A$, then B is valid. Thus positive invariance, while closely related to validity, is a different concept.
 2. If A is α -valid at s and B is α -valid at s , then $A \cap B$ is α -valid at s . Note that $A \cap B$ is non-empty since it must contain $y(s)$.
 3. If A is valid at s and B is valid at s , $A \cap B$ need not be valid at s .

Proof. (of part 3) A could be α_1 -valid, B α_2 -valid, with $\alpha_1 \neq \alpha_2$. Just consider any point for the system in Example 6.

⁴ Implementability is an important concept. It refers not only to existence of a policy that could be physically carried out, but also to the fact that we will be able to actually compute and invoke it.

3.2 Servo Controllers

Now, we are ready to describe the functions of our three-level hierarchical control. Abstractly, there are two ways to view the servo controller:

- As receiving from the reflex controller a policy (or “code” for it) which it must enforce.
- As receiving from the reflex controller a set (or some description or “code” for it) which it must keep invariant.

As noted before, herein we think of the servo controller as receiving “commands” in the form of bounding sets called *setpoints* from the reflex controller. Thus, we assume that the servo controller, given a valid setpoint $S(t_i)$, acts in such a manner as to contain the system in that set. More specifically, it has the property:

Condition 1 (Quasi-No-Overshoot, Viable) 1. *Given a valid bounding set at time t_i , $S(t_i)$, the system and at least one of its possible sweep volumes will stay within the bounding set until a new bounding set is approved, i.e.,*

$$\text{for all } t_{i+1} \geq t \geq t_i, \text{ there exists } \alpha \in P \text{ such that } B_\alpha(s(t)) \subset S(t_i),$$

where t_{i+1} is the time a new bounding set is approved.

2. *Given an α -valid bounding set at time t_i , $S(t_i)$, the system and at its α sweep volume will stay within the bounding set until a new bounding set is approved, i.e.,*

$$B_\alpha(s(t)) \subset S(t_i), \quad \text{for all } t_{i+1} \geq t \geq t_i$$

where t_{i+1} is the time a new bounding set is approved.

In such as case, we say that $S(t_i)$ is viable (α -viable).⁵

Note: In the present work, the time t_{i+1} is not known in advance and could be infinite.

Example 7. Suppose we invoke the linear PD law of Equation (4) with K_d and K_v chosen to be critically- or over-damped. Suppose further that from (x_0, v_0) we can stop at $x_d = x_b$ without overshoot. Then, one can show that we may stop without overshoot using any point $x_d = x_b + \text{sgn}(x_b - x_0)\beta^2$.

The idea of quasi-no-overshoot is simply that there exist implementable control inputs such that the bounding set is positively invariant with respect to the property of being valid: once valid, (can be kept) always valid. Since the servo need only invoke the valid policy to insure this, this condition is feasible. We allow the servo more freedom than this, though; we allow it to “use” as much of the setpoint set as it wants, as long as it can guarantee that it does not exit it. The situation is thus akin to the two steps of solving a linear program: the reflex controller proves the constraints are feasible—and perhaps gives a feasible point. The servo may, however, pick any other *feasible* point as a solution.

⁵ Cf. viability theory [1].

3.3 Reflex Controller

Keeping with our abstraction, we take the view that the reflex controller receives requests for bounding sets from higher-level processes. It then decides whether to approve these requests based on the system’s state and the presence of constraints. The reflex controller may then:

1. Deny the request outright because it is not valid, leaving the servo controller with its last bounding set.
2. Approve the requested set.
3. Approve a subset of the requested set.

To achieve constraint satisfaction, then, one merely needs to give to the servo controller valid bounding sets which are violation-free: This leads immediately to guaranteed constraint satisfaction if

Condition 2 (Obstacle Avoidance, α -Obstacle Avoidance) *The reflex controller only approves valid (α -valid) bounding sets which are violation-free.*

Finally, we combine these two properties into what we term the Obstacle-Avoidance-State:

Condition 3 (Obstacle-Avoidance-State) *The system is operating in a state consistent with constraint satisfaction if $S(t_i)$ is viable (α -viable) and violation-free for all $t_{i+1} \geq t \geq t_i$. Here, t_{i+1} is the next time that the setpoint set is updated.*

For well-posedness of the obstacle-avoidance problem, we require that the system is initially in the Obstacle-Avoidance-State (at $i = 0$). Given this, the following is an update procedure which guarantees obstacle avoidance for all time based on keeping track of only one control policy at a time:

Algorithm 1 (α -Update) *Assume we are in the α -Obstacle-Avoidance-State.*

0. If a new request is received, go to Step 1. Otherwise go to Step 0.
1. At time T ($t_{i+1} \geq T \geq t_i$), the reflex controller receives a request set $R(T)$.
 - (a) If $R(T)$ is not α -valid (at $s(T)$) we do not update the setpoint and continue to operate in the Obstacle-Avoidance-State. Go to Step 0.
 - (b) If it is found to be α -valid, we send $S(T)$ to the servo controller where

$$S(T) = S(t_i) \cap R(T)$$

Note that this set is α -valid because it is the intersection of two α -valid sets. It is violation-free because $S(t_i)$ is violation-free.⁶ We set $t_{i+1} = T$ and increase i in the Obstacle-Avoidance-State. Go to Step 2.

2. Now, we check the set $R(T)$ (actually, we need only search the set $R(T) - S(T)$) for obstacles. Let us assume that the inspection task is completed at time $t_{i+1} \geq T$.

⁶ Note that we have also assumed that the check for α -validity is instantaneous. We can relax this condition.

(a) If the request set is violation-free, then we approve it: Set

$$S(t_{i+1}) = R(T),$$

increase i , go back to the α -Obstacle-Avoidance-State, and go to Step 0.

(b) If it has obstacles, then we approve instead some violation-free set, $S(t_{i+1})$, such that

$$S(T) \subset S(t_{i+1}) \subset R(T)$$

increase i , and go back to the Obstacle-Avoidance-State. Such a set is guaranteed to exist because $S(T) \subset R(T)$ and $S(T)$ is violation-free. It is α -valid because it is a superset of the α -valid set $S(T)$. Go to Step 0.

Note that we may switch the control policy in the above algorithm from α to α' whenever $S(t_i)$ is α' -valid. The following algorithm allows updates without reference to a specific control policy:

Algorithm 2 (Update) Assume we are in the Obstacle-Avoidance-State.

0. If a new request is received, go to Step 1. Otherwise go to Step 0.
1. At time T ($t_{i+1} \geq T \geq t_i$), the reflex controller receives a request set $R(T)$.
 - (a) If $S(T) = R(T) \cap S(t_i)$ is not valid we do not update the setpoint and continue to operate in the Obstacle-Avoidance-State. Go to Step 0.
 - (b) If it is found to be valid, we send $S(T)$ to the servo controller. It is violation-free because $S(t_i)$ is violation-free. We set $t_{i+1} = T$ and increase i in the Obstacle-Avoidance-State. Go to Step 2.
2. [Same as Step 2 for α -Update, but with “valid” replacing “ α -valid.”]

Note: The two-step approval process in the algorithms above is used because of the computation time required to check if a region is obstacle-free.

4 Analysis of Robotic Collision Avoidance

The control commands consist of prisms replacing abstract sets for sweep (hereafter, braking), request, and setpoint above. We assume decoupled joints as in Equation (3), a no-overshoot servo controller in each joint as in Equation (4), and a point-to-point reflex implementation.

We would like the robot to go (if possible) to a requested goal point g without colliding with any obstacles. In order to maintain constraint satisfaction (viz., obstacle avoidance in this example), the active reflex generates setpoints. In some cases, the reflex controller may approve the goal outright. In most cases, however, the setpoints it generates will be intermediate (as described below) and we call such points *subgoals*.

4.1 Subgoals and Convergence of Subgoals

Let's discuss in detail one particular algorithm for picking subgoals. Consider the hyperprism, $[q(t), g]$ obtained by the current configuration $q(t)$ and the current goal g . Similarly, consider the braking volume, $[q(t), b(t)]$, and maximum acceleration prism, $[q(t), a(t)]$, where $b(t)$ and $a(t)$ were defined in Section 2.4. Finally, consider the hyperprism $[q(t), o(t, p)]$, where $o(t)$ is the "closest" obstacle point in the "direction" of g obtained by *extending from the prism* $[q(t), p]$. To make things rigorous, we add joint limits as obstacles.

Note: *we only consider the case where we are "heading in the direction of the goal" in all joints*, in terms of the velocities $\dot{q}(t)$. If this were not the case, we could expand the request prism to include the braking volume. We also assume that $[q(t_0), b(t_0)] \subset [q(t_0), g] \cap [q(t_0), o(t_0, b(t_0))] \cap [q(t_0), a(t_0)]$ for well-posedness of the obstacle-avoidance task (that is, we begin in the Obstacle-Avoidance-State).

We update subgoals as follows. Given that we are heading in the direction of the goal in all joints, it makes sense that subgoals are always chosen to be within $[q(t), g]$. With all our notation, the k th subgoal, $s(t_k) = s^k$, chosen at time $t = t_k$ is simply the point in \mathbf{R}^n such that

$$[q(t), s^k] = [q(t), g] \cap [q(t), o(t, s^{k-1})] \cap [q(t), a(t)], \quad (5)$$

where s^{k-1} is the previous subgoal. A different, useful scheme is to pick

$$[q(t), s^k] = [q(t), g] \cap [q(t), o(t, b(t))] \cap [q(t), a(t)], \quad (6)$$

but the convergence analysis requires more assumptions. It is clear that the update law of Equation (5) is such

$$[q(t), b(t)] \subset [q(t), s(t)], \quad \text{for all } t \geq t_0. \quad (7)$$

4.2 The Discretized Analysis

Let the coordinates of the robot at time t be given by the n -tuple $q(t) = (q_1(t), \dots, q_n(t))$. Similarly, let the current goal of the robot be given by the n -tuple $g = (g_1, \dots, g_n)$. Finally, let the sequence of subgoals of the robot be given by the n -tuple $s^k = (s_1^k, \dots, s_n^k)$. The distance from subgoal to goal may be computed with any suitable norm on \mathbf{R}^n , e.g., $\|s^k - g\|_1$, $\|s^k - g\|_2$, $\|s^k - g\|_\infty$.

Our above algorithm is such that $s_i^{k+1} \in [s_i^k, g]$. Thus, our algorithm for picking subgoals is such that $|s_i^{k+1} - g_i| \leq |s_i^k - g_i|$; it also has the property that $s_i^{k+1} - g_i$ and $s_i^k - g_i$ are both either non-negative or non-positive. Note that this immediately implies $\|s^{k+1} - g\| \leq \|s^k - g\|$, for the example norms

above. Indeed, it holds for any l_p - or weighted l_p -norm.⁷ Finally, let us consider the sequence of subgoals $\{s^j\}$, and its component sequences, $\{s_i^j\}$, $i = 1, \dots, n$. Each component sequence is monotonically converging to g_i , a finite number. Elementary analysis [18, Thm 3.14, p. 55] gives $\{s_i^j\}$ converges to some s_i , $i = 1, \dots, n$. This componentwise convergence implies that $\{s^j\}$ converges to $s = (s_1, \dots, s_n)$, the convergence holding for any \mathbf{R}^n norm [19, Fact 3.1(56), p. 58].

4.3 The Continuous Analysis

This is immediate. The update rules and PD control law ensure that we are viable for each subgoal. The PD control law further ensures convergence towards each subgoal.

5 Conclusions and Future Work

We have presented a means to achieve guaranteed constraint satisfaction of a hybrid dynamical system (which takes into account the underlying continuous dynamics) in a simple, hierarchical control algorithm. Two layers of functionality, (1) piece-wise viable servo controllers and (2) a “reflex controller,” are required for guaranteed constraint satisfaction.

The philosophy has been successfully applied to a robot’s maintaining collision avoidance while under higher-level control, viz. a variety of control algorithms, path planners, and teleoperation. [16, §7]. In [9], a means to string goals together in order to accomplish global planning (with local obstacle avoidance guaranteed by the reflex and servo layers) is presented. Basically, graph search on “dynamically visible” points is used.

In future work, we will examine different ways to combine constituent dynamical systems (control policies) together in order to achieve higher-level goals. Finally, the reflex concept presented herein should be just as effective in an on-line environment using bounding polytopes (given by linear inequalities) or ellipses. In these cases, the speed and efficacy of linear programming and linear matrix inequalities (LMIs) should prove advantageous.

⁷ Note that it does not hold for all \mathbf{R}^n norms: consider the counterexample $x = [1, 1]^T$, $y = [1, -2]^T$ and the basis vectors $b_1 = [1, -1]^T$, $b_2 = [-1, 2]^T$. In this basis, $x = 3b_1 + 2b_2$, $y = -b_2$, so that

$$\|x\|_{\mathcal{B}, \max} = 3 > 1 = \|y\|_{\mathcal{B}, \max},$$

where $\|\cdot\|_{\mathcal{B}, \max}$ is the $\|\cdot\|_{\infty}$ norm applied to the coefficients in the new basis. See [8, pp. 77–78] for the validity of $\|\cdot\|_{\mathcal{B}, \max}$ as a norm. It does not even hold when x_i and y_i are constrained to both be non-negative (or non-positive) for each i : consider the counterexample $x = [1, 1]^T$, $y = [1, 2]^T$ and the basis vectors $b_1 = [-1, 3]^T$, $b_2 = [1, 2]^T$. In this basis, $x = b_1 + 2b_2$, $y = b_2$, so that

$$\|x\|_{\mathcal{B}, \max} = 2 > 1 = \|y\|_{\mathcal{B}, \max}.$$

References

1. Jean-Pierre Aubin. *Viability Theory*. Birkhauser, Boston, 1991.
2. Anthony J. Barbera, M. L. Fitzgerald, and J. S. Albus. Concepts for a real-time sensory-interactive control system architecture. In *Proc. Fourteenth Southeastern Symp. on System Theory*, pp. 121–126, April 1982.
3. Michael S. Branicky. Efficient configuration space transforms for real-time robotic reflexes. Master's thesis, Case Western Reserve University, Dept. of Electrical Engineering and Applied Physics, January 1990.
4. Michael S. Branicky. Analyzing continuous switching systems: theory and examples. In *Proc. American Control Conf.*, pp. 3110–3114, Baltimore, June 1994.
5. Michael S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, June 1995.
6. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.
7. Yao-Chon Chen and Mathukumalli Vidyasagar. Optimal trajectory planning for planar n -link revolute manipulators in the presence of obstacles. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pp. 202–208, 1988.
8. Morris W. Hirsch and Stephen Smale. *Differential Equations, Dynamical Systems, and Linear Algebra*. Academic Press, Inc., San Diego, CA, 1974.
9. Vinay K. Krishnaswamy and Wyatt S. Newman. On-line motion planning using critical point graphs in two-dimensional configuration space. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pp. 2334–2340, Nice, France, May, 1992.
10. Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.
11. Tomás Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Trans. Systems, Man, and Cybernetics*, SMC-11(10):681–698, October 1981.
12. Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Computers*, C-32(2):108–120, February 1983.
13. Wyatt S. Newman. High speed robot control and obstacle avoidance using dynamic potential functions. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pp. 14–24, March 1987.
14. Wyatt S. Newman. *High-Speed Robot Control in Complex Environments*. PhD thesis, Massachusetts Institute of Technology, Dept. of Mechanical Engineering, October 1987.
15. Wyatt S. Newman. Automatic obstacle avoidance at high speeds via reflex control. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pp. 1104–1109, May 1989.
16. Wyatt S. Newman and Michael S. Branicky. Experiments in reflex control for industrial manipulators. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pp. 266–271, May 1990.
17. Amir Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Foundations of Computer Science*, Providence, RI, pp. 46–57, October 31–November 2, 1977.
18. Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, New York, third edition, 1976.
19. M. Vidyasagar. *Nonlinear Systems Analysis*. Prentice-Hall, Englewood Cliffs, 1978.