Noname manuscript No.
(will be inserted by the editor)

Hybrid Systems Tools for Compiling Controllers for Cyber-Physical Systems

Patrick Martin · Magnus Egerstedt

Abstract In this paper, we consider the problem of going from high-level specifications of complex control tasks for cyber-physical systems to their actual implementation and execution on physical devices. This transition between abstraction levels inevitably results in a specification-to-execution gap, and we discuss two sources for this gap; namely model based and constraint based. For both of these two types of sources, we show how hybrid control techniques provide the tools needed to compile high-level control programs in such a way that the specification-to-execution gap is removed. The solutions involve introducing new control modes into nominal strings of control modes as well as adjusting the control modes themselves.

1 Introduction

Many emerging controls applications have seen increased system complexity due to the pervasive use of computing and networking resources [1,14] in tight coupling with the physical world in which these devices are deployed. The resulting, so-called *cyber-physical* systems (CPS), have been deployed to instrument and control large scale systems, from buildings to sensor networks. When designing controllers for such systems it is no longer feasible to design one control law that is expected to cope with all possible environmental conditions the system can be experience. As such, the control laws have to be *hybrid* in nature. In fact, in order to manage this complexity, system abstractions and specification languages must be designed so that control tasks can be broken up into smaller "chunks," or motion primitives, which are then interpreted by the systems at run-time.

Patrick Martin
York College of Pennsylvania
York, PA 17403, USA
E-mail: pjmartin@ycp.edu

Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA
E-mail: magnus@gatech.edu

One aspect of this abstraction-based approach to control design is that it lets users specify a desired system execution using strings of pre-defined control laws. However, these types of abstractions typically lose some of the important physical details that affect the *execution* of the control laws. This *specification-to-execution* gap can result in undesirable behavior at run-time. It would be beneficial if there existed an intermediate process, such as the one illustrated in Figure 1, that takes a system specification and brings it in-line with system capabilities and constraints before execution.

The problem under consideration in this paper is that of designing this intermediate process. As will be seen, we will not provide a one-size-fits-all solution, but rather discuss some of the issues that must be addressed and, in particular, show how hybrid systems theory provides valuable tools for this endeavor. It should be noted, already at this point, that some of the technical results presented in this paper have appeared elsewhere, and the real novelty of the paper must be understood in terms of the placement of these results in the context of compilation of control programs for CPS.

The outline of this paper is as follows: In Section 2, we further discuss the specification-to-execution gap and show why the problems associated with this are inherently hybrid in nature. In particular, we formulate the specifications for cyber-physical systems in terms of motion description languages – strings of controller-interrupt pairs – to be parsed by the system. We also identify the two key aspects of the compilation process that will be addressed in this paper, namely the problem of modifying the particulars of the control laws and interrupt conditions to make them fit the physical platform better, and the problem of inserting new control modes in order to satisfy constraints imposed on the controller by the physical platform. The latter of these problems is discussed in Section 3, where we introduce hard control constraints and show how a hybrid control design process will allow us to insert new control modes into the specifications in an automatic fashion in order to satisfy the otherwise violated constraints. In Section 4, recent results in optimal timing control for switched systems are used to modify the control strings prior to their execution. This approach is illustrated on a problem concerning how to control a robotic marionette based on high-level specifications. A discussion of where this line of inquiry should be taken next is given in Section 5.

2 The Specification-to-Execution Gap

One challenge associated with the intermediary compilation process is the inherent heterogeneity of CPS due to their varying computational capabilities. Additionally, actuators and sensors may be different among types of a single class of systems, such as a collection of mobile robots. These robots may have the same drive train and the physical dynamics; however, each robot may be equipped with different suites of sensors from IR to GPS. The selection of sensors affects the ability of the robot to execute control tasks. Despite this, it is desirable to specify, at a high level of abstraction, what the different systems should be doing without having to take such low-level details into account. Consequently, new specification languages are required to enable the specification of control tasks among systems with different capabilities.

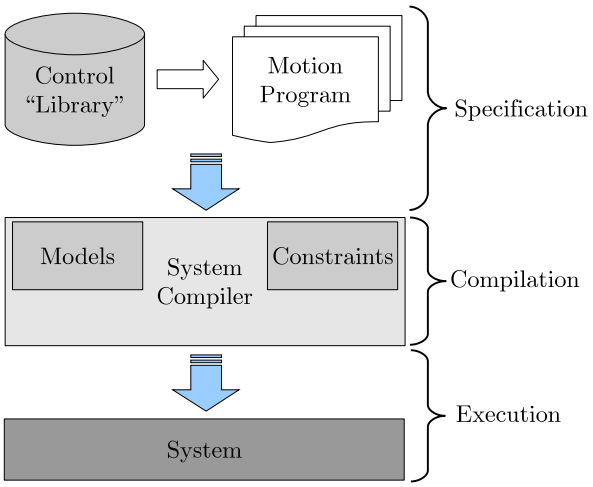


Fig. 1 This work focuses on developing the tools necessary for the development of an intermediate, or "compilation," layer for creating motion programs for cyber-physical systems.

2.1 Motion Description Languages

For CPS, one design methodology that has proven useful is to decompose the control task into sequences of primitive building blocks, designed with a particular objective in mind. These building blocks are then sequenced together to produce the desired, overall system behavior. One way in which this sequencing of motion programs can be done is in the framework of Motion Description Languages (MDL), consisting of strings of controller-interrupt pairs, e.g. [3,9,11]. For example, in mobile robotics, one can construct motion programs with MDL that define a sequence of behaviors for navigation through an environment.

More specifically, let the system have dynamics of the form

$$\dot{x} = f(x,u),\ x \in \mathcal{X} \subseteq \mathbb{R}^n, u \in \mathcal{U} \tag{1}$$

where x is the state of the system, and u is the control input. If we let the control input be given by a closed-loop mapping $\kappa : \mathcal{X} \to \mathcal{U}$, and introduce interrupt functions as mappings $\xi : \mathcal{X} \to \{0,1\}$, where 1 indicates that an interrupt has occurred, we denote a MDL mode by the tuple (κ, ξ). The interpretation here is that the system in (1) executes the controller κ until $\xi \to 1$. A MDL program thus becomes a sequence of such controller-interrupt pairs.

We can think of the physical system as parsing strings in the MDL. For example, given the MDL string $(\kappa_1, \xi_1), (\kappa_2, \xi_2), \ldots$ the system parses this string as

$$\dot{x} = f(x, \kappa_1(x)), \quad \text{until} \quad \xi_1(x) = 1,$$

at which point

$$\dot{x} = f(x, \kappa_2(x)), \quad \text{until} \quad \xi_2(x) = 1,$$

and so forth. This way of specifying controllers has been used in robotics [9], manufacturing [3], and mobile sensor networks [15], and we take it as a prototypical specification language for CPS, even though it should be noted that other choices are possible, such

as LTL [5], Maneuver Automata [7], CCL [8], and Control Quanta [2], just to name a few.

2.2 Compiling MDL for Cyber-physical Systems

Regardless of specification language, it is generally assumed that the controller "fits" the dynamics of the system in that the system can execute the control string and that the transitions between control modes are well-behaved. However, in a number of practical applications, e.g. embedded control systems, there are hard constraints on the actuator signals achievable that effect what motion programs can be executed. It is thus possible that a motion program that is intended to perform some action, actually fails to accomplish the task because of these constraints. Moreover, unmodeled dynamics or simply unforeseen environmental conditions may cause the transitions between modes to occur at the wrong time or in the wrong manner. As such, we identify two distinct, different sources for the specification-to-execution gap, namely *constraint-based* and *specification-based* sources.

3 Constraint-Based Compilation

As a canonical example of the issue with constraints associated with the actual platform on which the CPS program is deployed, we consider the effect that input bounds have on the system. A preliminary treatment of this problem was given in [10].

Consider the unstable, controllable linear system

$$\dot{x} = Ax + bu,\ x \in \mathbb{R}^n,\ u \in \mathbb{R}. \tag{2}$$

where $x \in \mathbb{R}^n$ and $u \in \mathbb{R}$, and A, b are matrices of appropriate dimension. A stabilizing controller for this system can be generated by solving the LQ design problem where the cost functional is

$$J(x,u) = \int_0^\infty \left(x^T Q x + \sqrt{2} u^T u \right) dt.$$

The solution is given by the feedback gain P that solves the Riccati equation

$$A^T P + PA + Q - 2Pbb^T P = 0, \tag{3}$$

where $Q \succ 0$. The resulting feedback control for stabilizing (2) is thus given by

$$u = -b^T P x. \tag{4}$$

What we will do is select such a P and then augment the controller with an affine term that drives the system to a given set-point. By stringing together different such affine terms, we get a motion program that takes the system through a sequence of set-points. Consequently, we modify the control input to include an affine term, v, as

$$u = -b^T P x + v. \tag{5}$$

By applying the controller (5) to (2), we get the following closed loop dynamics:

$$\dot{x} = (A - bb^T P)x + bv,$$

with globally attractive, stationary point given by

$$x_v = -(A - bb^T P)^{-1} bv.$$

(Note that the inverse is well defined since the real parts of the eigenvalues of $(A - bb^T P)$ are negative by design.) As such, we can define the interrupt function $\xi(P, v, \epsilon)$ that triggers once the state of the system is close to x_v, i.e.

$$\xi(P, v, \epsilon) = \begin{cases} 1 & \text{if } \|x - x_v\|_P^2 \leq \epsilon \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where $\|z\|_P^2 = z^T P z$.

As such, we let the motion program used for driving the system through a collection of set-points be given by strings $((P, v_1), \xi(P, v_1, \epsilon_1)), \ldots, ((P, v_N), \xi(P, v_N, \epsilon_N))$. As we have assumed that P is fixed, we can use the shorthand $(v_1, \epsilon_1), \ldots, (v_N, \epsilon_N)$ to denote these MDL strings.

For each $v \in \mathbb{R}$, we thus have a controller that takes the system (asymptotically) to the set-point x_v. The first thing we need to do in order to "compile" the MDL programs is to characterize what the set of such set-points actually looks like under the input constraint $|u| < u_{max}$.

3.1 Regions of Attraction

Let the set of stationary points be denoted by $\mathcal{X}$. In order to calculate the regions of attraction around each point, we need the following lemma:

Lemma 1 (Stationary Points under Bounded Input) *Let (A, b) be a controllable pair, let P be the positive definite matrix solution to the Riccati equation* (3) *for some $Q \succ 0$, and let $K = (A - bb^T P)$. If $u = -b^T Px + v$ and $|u| < u_{max}$, then the set of stationary points $\mathcal{X}$ is given by*

$$\mathcal{X} = -K^{-1} b (b^T P K^{-1} b + 1)^{-1} [-u_{max}, u_{max}]$$

if $b^T P K^{-1} b + 1 \neq 0$ or,

$$\mathcal{X} = -K^{-1} b \mathbb{R}, \text{ otherwise.}$$

Proof Assume that $x_{v_i} \in \mathcal{X}$ is the stationary point obtained by using the open-loop control v_i in equation (5). Then, the total control effort needed to hold the system at x_{v_i} is

$$\begin{aligned} u_{v_i} &= -b^T P x_{v_i} + v_i \\ &= (b^T P K^{-1} b + 1) v_i \end{aligned}$$

Note that if $b^T P K^{-1} b + 1 = 0$ then $u_{v_i} = 0$ for any $v \in \mathbb{R}$. If the equality *does not* hold, then the choice of v must come from the set

$$v \in \mathcal{V} = (b^T P K^{-1} b + 1)^{-1} [-u_{max}, u_{max}].$$

in order to maintain that $|u| < u_{max}$; otherwise, $v \in \mathbb{R}$. Hence, the set of stationary points is

$$\mathcal{X} = -K^{-1} b \mathcal{V}$$

which completes the proof.

Now, with a proper characterization of these stationary points, we can proceed with calculating the regions of the state space from which these points can be reached with bounded inputs. These regions will form the basis for developments in the following sections.

Theorem 1 (Regions of Attraction Under Bounded Input) *Given the assumptions in Lemma 1, the region of attraction around the point x_{v_i} is given by*

$$\mathcal{E}(P, v_i) = \{x \in \mathbb{R}^n | (x - x_{v_i})^T P (x - x_{v_i}) \leqslant \alpha_{v_i}\} \tag{7}$$

with

$$x_{v_i} = -K^T b v_i$$

and

$$\alpha_{v_i} = (b^T P b)^{-1} (u_{max} - |u_{v_i}|)^2. \tag{8}$$

Proof Let $x = x_{v_i} + \Delta x_{v_i}$, then

$$u = -b^T P (x_{v_i} + \Delta x_{v_i}) = u_{v_i} - b^T P \Delta x_{v_i}.$$

However, since we have the constraint $u \in [-u_{max}, u_{max}]$, we interpret the above equation as

$$b^T P \Delta x_{v_i} \in [-u_{max} + u_{v_i}, u_{max} + u_{v_i}].$$

P is a solution to the Riccati equation (3), so we define the function

$$V(\Delta x_{v_i}) = \Delta x_{v_i}^T P \Delta x_{v_i},\ \forall \Delta x_{v_i} \in \mathbb{R}^n,\ \Delta x_{v_i} \neq 0. \tag{9}$$

If this function is Lyapunov, then it can serve as a conservative region of attraction around the point x_{v_i}. To show this fact, consider the ellipsoid generated by $\Delta x_{v_i}^T P \Delta x_{v_i} = \gamma$, where $\gamma > 0$ and real and $\forall \Delta x_{v_i} \in \mathbb{R}^n$. Assume γ is chosen such that $| - b^T P \Delta x_{v_i}| < u_{max}$, and, furthermore, we choose some $\beta \in (0, \gamma)$.

Thus, we want to solve the following maximization problem for any $\Delta x_{v_i} \in \mathbb{R}^n$:

$$\begin{aligned} &\max\nolimits_{\Delta x_{v_i}} b^T P \Delta x_{v_i} \\ &\text{s.t.}\ \ \Delta x_{v_i}^T P \Delta x_{v_i} = \gamma. \end{aligned}$$

Forming the Lagrangian,

$$L = b^T P \Delta x_{v_i} - \lambda (\Delta x_{v_i}^T P \Delta x_{v_i} - \gamma),$$

and setting its derivative w.r.t. Δx_{v_i} equal to 0 we get that

$$\Delta x_{v_i} = -\frac{1}{2\lambda} b \tag{10}$$

which implies that the maximum Δx_{v_i} is parallel to the input matrix b. According to the constraint equation we calculate the value of λ and insert into equation (10), resulting in the maximum value:

$$\Delta x_{max} = \sqrt{\frac{\gamma}{b^T P b}} b.$$

Using this value in the control law results in the maximum control effort

$$u_\gamma = b^T P \sqrt{\frac{\gamma}{b^T P b}} b = \sqrt{\gamma} \|b\|_P$$

Finally, if we compare the difference between the maximum input along the the γ level-set and that on the β level-set we get

$$u_\beta - u_\gamma = (\sqrt{\beta} - \sqrt{\gamma}) \|b\|_P$$

which is a strictly negative quantity, by the assumption that $\beta \in (0, \gamma)$. Therefore, the control effort reduces as the state decays within the ellipsoid $\Delta x_{v_i}^T P \Delta x_{v_i}$, and (9) is a Lyapunov equation.

Now that we know the region defined by (9) is Lyapunov, we find the solution to

$$\min_{\Delta x_{v_i}} \Delta x_{v_i}^T P x_{v_i}$$

such that

$$b^T P \Delta x_{v_i} = -u_{max} + u_{v_i}$$

or

$$b^T P \Delta x_{v_i} = u_{max} + u_{v_i}.$$

Letting $c = Pb$ and $d_\pm = \pm u_{max} + u_{v_i}$, the Lagrange necessary and sufficient conditions (see for example [**?**]) for these quadratic optimization problems are:

$$\begin{aligned} P \Delta x_{v_i} + c \lambda_{v_i} &= 0 \\ c^T \Delta x_{v_i} - d_\pm &= 0 \end{aligned}$$

where $\lambda_{v_i} \in \mathbb{R}^n$ is the Lagrange multiplier. Solving these equations results in

$$\begin{aligned} \lambda_{v_i} &= -(c^T P^{-1} c)^{-1} d_\pm \\ \Delta x_{v_i} &= P^{-1} c (c^T P^{-1} c)^{-1} d_\pm. \end{aligned}$$

Inserting the solution for Δx_{v_i} into (9) results in:

$$\Delta x_{v_i}^T P \Delta x_{v_i} = (b^T P b)^{-1} (\pm u_{max} + u_{v_i})^2$$

where $(b^T P b)^{-1}$ exists since $P \succ 0$ and $b \neq 0$ by our controllability assumption. We choose the smallest and define it as

$$\alpha_{v_i} = (b^T P b)^{-1} (u_{max} - |u_{v_i}|)^2.$$

Therefore, the region around each stationary point x_{v_i} where $|u| \leq u_{max}$ is given by

$$\mathcal{E}(P, v) = \{x \in \mathbb{R}^n | (x - x_{v_i})^T P (x - x_{v_i}) \leq \alpha_{v_i}\},$$

as in equation (7), and the theorem follows.

A direct corollary of Theorem 1 follows that describes the characterization of the entire region of attraction about the points in $\mathcal{X}$:

Corollary 1 *The region of attraction around the set of stationary points $\mathcal{X}$ is given by*

$$\mathcal{L} = \bigcup_{v_i \in \mathcal{V}} \mathcal{E}(P, v_i).$$

3.2 Checking for Feasibility

We are interested in determining whether a given MDL program $(v_1, \epsilon_1), \ldots, (v_N, \epsilon_N)$ will successfully drive the system between the desired set-points under the input constraint $|u| \leq u_{max}$. It is clear that v_i must be in $\mathcal{V}$ for this to be possible, so we first make the following assumption:

Assumption 1 $v_i \in \mathcal{V},\ i = 1, \ldots, N.$

Now, in order for a MDL string to be feasible, the i^{th} set-point must lie in region of attraction for the $(i+1)^{th}$ set-point, and we state this fact as a lemma:

Lemma 2 *Given an MDL string*

$$\sigma = (v_1, \epsilon_1), \ldots, (v_N, \epsilon_N),$$

satisfying Assumption 1, let

$$\mathcal{B}_{\epsilon_i}(v_i) = \{x \in \mathbb{R}^n \mid (x - x_{v_i})^T P (x - x_{v_i}) \leq \epsilon_i\}$$

represent an ellipse around point x_{v_i}. If

$$\mathcal{B}_{\epsilon_i}(v_i) \subseteq \mathcal{E}(P, v_{i+1}) \tag{11}$$

then x can be transferred from x_{v_i} to $x_{v_{i+1}}$ while satisfying the bounded input constraint.

(Note here that the set $\mathcal{B}_{\epsilon_i}(v_i)$ is exactly the set where interrupt in equation (6) takes on the value 1.)

This lemma states that if the ellipse of size ϵ_i around point x_{v_i} is strictly within the region of attraction of $x_{v_{i+1}}$, then the system will arrive at $x_{v_{i+1}}$ (asymptotically) and satisfy the input constraints. Based on this pairwise characterization of the feasibility, we can now extend this notion to entire MDL strings:

Assumption 2

$$x(0) \in \mathcal{E}(P, v_1).$$

Definition 1 *The string*

$$\sigma = (v_1, \epsilon_1), \ldots, (v_N, \epsilon_N),$$

is a feasible program string *if it satisfies Assumptions 1 and 2, and*

$$\mathcal{B}_{\epsilon_i}(v_i) \subseteq \mathcal{E}(P, v_{i+1}),\ \textit{for } i = 1, \cdots, N-1.$$

The set of these feasible program strings is denoted by $\mathcal{F}$.

We state the following theorem (whose rather obvious proof we omit):

Theorem 2 (Program feasibility) *The MDL string σ drives the system close to the set-points (in the sense defined by the interrupts) if $\sigma \in \mathcal{F}$.*

What this theorem gives us is a feasibility check for determining if the string does in fact perform as desired. However, if the feasibility condition is violated[1], something must be done, and in the next section, we discuss how to insert new control modes into the MDL string in order to respect the bounded input constraints yet drive through the desired intermediary set-points.

[1] Since our Lyapunov functions are *conservative* estimates of the region of attraction around each point, it may still be possible to execute an MDL string even if $\sigma \notin \mathcal{F}$.

3.3 Mode Insertion to Maintain Input Bounds

When an MDL string fails the feasibility check, we need to modify the string to ensure that the input constraints are satisfied. Our approach is to insert new modes into the string σ, so that the augmented string becomes a member of the feasible set $\mathcal{F}$. This method was inspired by *sequential composition*, as described in [4], by inserting modes when we know that the inserted region of attraction contains the a subset of the region of attraction of the previous mode.

Consider, again, the MDL string

$$\sigma = (v_1, \epsilon_1), \ldots, (v_N, \epsilon_N).$$

Each element in σ comes from a finite alphabet of modes, which we denote by $(v_i, \epsilon_i) \in \mathcal{A}$. The set of all possible concatenations of these elements is denoted by $\mathcal{A}^\star$; consequently, the each MDL comes from the set of all concatenations, i.e. $\sigma \in \mathcal{A}^\star$. We define the length operator as a mapping $\ell : \mathcal{A}^\star \to \mathbb{N}$, which accepts an MDL string and returns its number of elements. For example, if $\sigma = (v_1, \epsilon_1), (v_3, \epsilon_3)$, then $\ell(\sigma) = 2$.

Now, using our definitions, we state the mode insertion problem as:

$$\begin{aligned} &\min_{\sigma'} \ell(\sigma') \\ \text{s.t.} \quad &(v_i, \epsilon_i)\sigma'(v_{i+1}, \epsilon_{i+1}) \in \mathcal{F}. \end{aligned}$$

Thus, the problem of inserting intermediate points is characterized by minimizing the length of the string σ' such that the new string $(v_i, \epsilon_i)\sigma'(v_{i+1}, \epsilon_{i+1})$ is still in the set of feasible program strings, $\mathcal{F}$. This problem can be solved by inserting new modes, $(v_{k_l}, \epsilon_{k_l})$, such each pair of intermediate points satisfy the pairwise relation (11).

3.3.1 `MAXFORWARD` *Algorithm*

We develop an algorithm, called `MAXFORWARD`, that builds up the intermediate MDL string σ' using the relation in (11). This algorithm begins with the starting element of the MDL string: (v_i, ϵ_i). When the system uses this MDL mode, the state is pulled toward the point $x_{v_i} \in \mathcal{X}$ until it crosses into the interrupt region, $\mathcal{B}_{\epsilon_i}(v_i)$. From this region we need to insert a finite number of modes that can drive our system to $(v_{i+1}, \epsilon_{i+1})$.

We want to choose a v_{k_l} such that $\mathcal{B}_{\epsilon_i}(v_i) \subseteq \mathcal{E}(P, v_{k_l})$. In other words, we need an ellipse that covers the interrupt region so that equation (11) in Lemma 2 holds. To find the value of v_{k_l} that results in the covering ellipse $\mathcal{E}(P, v_{k_l})$, we design an iterative algorithm that steps through possible values of $v \in \mathcal{V}$, starting at v_i. At each iteration we perform the update

$$v_{k_{j+1}} = v_{k_j} + \Delta v,$$

where $\Delta v > 0$ is a fixed step size. As the size of Δv increases the more numerical error enters into the insertion process, eventually creating incorrect mode insertions. If this increment results in $\mathcal{B}_{\epsilon_i}(v_{k_j}) \nsubseteq \mathcal{E}(P, v_{k_{j+1}})$ then the value v_{k_j} is chosen as an intermediate MDL mode: $(v_{k_j}, \epsilon_{k_j})$. We repeat the algorithm until the m^{th} step, where

$$\mathcal{B}_{\epsilon_{k_m}}(v_{k_m}) \subseteq \mathcal{E}(P, v_{i+1}).$$

At this point we have the new intermediate string:

$$\sigma' = (v_{k_1}, \epsilon_{k_1}), \ldots, (v_{k_m}, \epsilon_{k_m}),$$

which maintains that $(v_i, \epsilon_i)\sigma'(v_{i+1}, \epsilon_{i+1}) \in \mathcal{F}$. Note that this algorithm produces an optimal path given our feasibility requirements. Without input bounds, the optimal solution would be more straightforward.

The formal algorithm is shown in listing Algorithm 1.

Algorithm 1 `MAXFORWARD` Algorithm

```
Choose Δv > 0
v_{k_j} ← v_i {Initialize with first MDL mode's controller.}
covered ← FALSE
while B_{ε_{k_j}}(v_{k_j}) ⊈ E(P, v_{i+1}) do
   while ¬covered do
      v_{k_{j+1}} = v_{k_j} + Δv
      if B_{ε_{k_j}}(v_{k_j}) ⊈ E(P, v_{k_{j+1}}) then
         σ' ← (v_{k_j}, ε_{k_j}) {Add interim MDL mode.}
         covered ← TRUE
      end if
   end while
end while
return σ'
```

Theorem 3 (`MAXFORWARD` **Optimality)**
If Algorithm 1 returns a solution σ' then it produces the minimal length string σ' such that $(v_i, \epsilon_i)\sigma'(v_{i+1}, \epsilon_{i+1}) \in \mathcal{F}$.

Proof Assume Algorithm 1 returned the string σ' corresponding to m intermediate points $x_{v_1}, \cdots, x_{v_m}$ between x_{v_i} and $x_{v_{i+1}}$, i.e. Algorithm 1 inserted m new modes into the MDL string. We note that in order to go from x_{v_k} to $x_{v_{i+1}}$ (or more precisely, from small ellipses around these points) Algorithm 1 needs $m - k + 1$ modes.

Now, since $v \in \mathbb{R}$, and hence $\dim(\mathbf{relint}(\mathcal{X})) = 1$, where **relint** denotes the relative interior, we can order points along $\mathcal{X}$ by how far away from $x_{v_{i+1}}$ they are. The first observation is that, by construction, the `MAXFORWARD` nature of Algorithm 1 prevents the existence of a point $x' \in \mathcal{X}$ such that x' can be reached in k or fewer steps from x_{v_i}, and $\|x' - x_{v_{i+1}}\| < \|x_{v_k} - x_{v_{i+1}}\|$. Instead, assume that σ' is *not* optimal and that the k^{th} intermediary point is $x'' \neq x_{v_k}$. Thus, we know that $\|x'' - x_{v_{i+1}}\| > \|x_{v_k} - x_{v_{i+1}}\|$.

But, the `MAXFORWARD` property again implies that no $x \in \mathcal{X}$ such that $\|x - x_{v_{i+1}}\| > \|x_{v_k} - x_{v_{i+1}}\|$ can reach $x_{v_{i+1}}$ in fewer steps than $m - k + 1$. As such, the optimal string (containing the intermediary point x'') can have no fewer elements than σ', and hence σ' is the (not necessarily unique) optimal solution.

This proof works because we have an order on the 1-dimensional space $\mathcal{X}$. If $\dim(\mathbf{relint}(\mathcal{X})) > 1$ then our algorithm and proof would have to consider multiple directions at each step due to the discretization of the state space.

3.3.2 Simulation Results

For our simulation results, we use the same choice of system matrices from the end of Section 3.1. We specified a two mode MDL string: $(v_1, \epsilon)(v_2, \epsilon)$, where each mode uses the same sized interrupt region defined by $\mathcal{B}_\epsilon(v_i),\ i = 1, 2$ and the values of v_i come from the computed region $\mathcal{V}$.

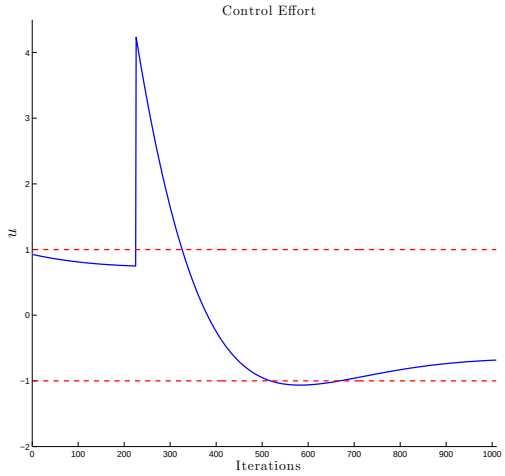


Fig. 2 The plot of the input u over the iterations of the simulation. Note that at the switch point, the system requires an input far greater than what the actuators can supply. The system leaves the input bound, again, as the second mode is executed.

Figure 2 shows the effort of the feedback controller as the system executes this MDL string. Once the system reaches the interrupt region of the first mode, the system switches to (v_2, ϵ), which causes a jump in the input signal that is well outside of the upper bound of the input constraint, $u_{max} = 1$. According to Definition 1, this MDL string is *not* feasible; hence, we must apply Algorithm 1 to insert intermediate modes.

The result of our `MAXFORWARD` algorithm, with $\Delta v = 0.001$, is shown in Figure 3 by the dotted ellipses. The algorithm inserted ten modes, allowing the system to move from its starting point x_0 to the final state x_f with bounded control effort, as shown in Figure 4. Using the expanded MDL string lengthens the time it takes for the system to approach x_f; however, our design goal of maintaining the input constraints was met.

4 Specification-Based Compilation

The second source of the specification-to-execution gap is, as already stated, caused by the fact that the control laws and interrupt conditions are specified at such a high level of abstraction that the actual dynamics of the system is not properly accounted for. For example, one would like to be able to let users define high-level tasks without having to worry about the actual dynamics. Yet, the dynamics clearly matters when it comes

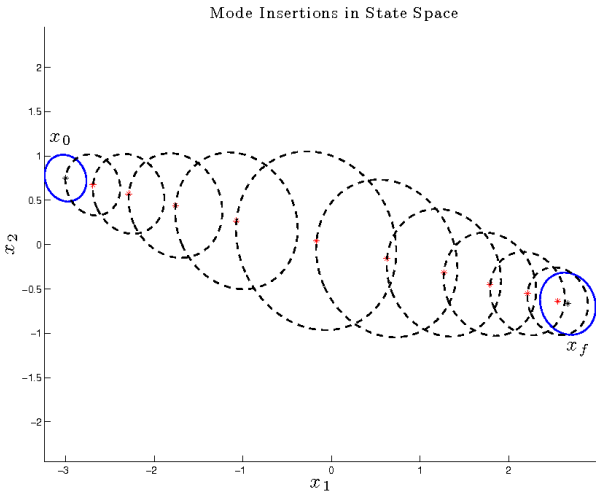


Fig. 3 This figure shows the ten new ellipses (dotted lines), $\mathcal{E}(P, v_{k_j})$ for $j = 1, \cdots, 10$, produced by the intermediate modes inserted into the MDL string.

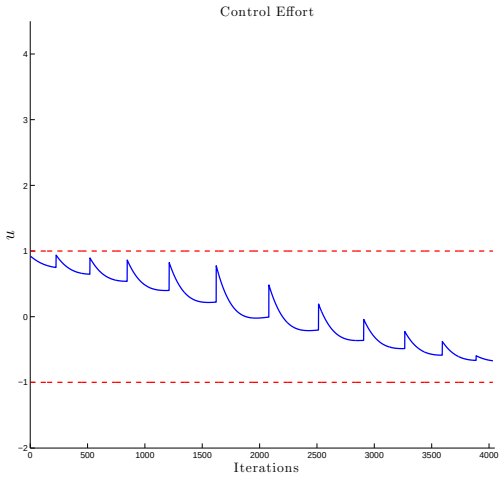
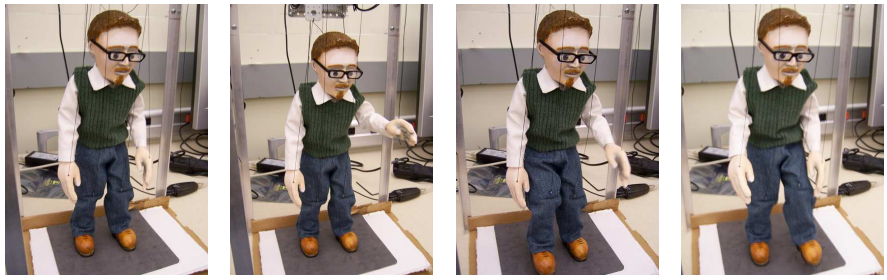


Fig. 4 In this plot, we see the successful maintenance of the control bound $|u| < 1$ during the execution of the expanded MDL string.

to executing these specifications. In this section, we illustrate this issue through an example involving robotic marionettes and, in particular, we show how recent results on optimal control of hybrid systems can by put to use as a compiler of CPS programs.

4.1 Example: Robotic Puppetry

Consider the problem of designing controllers that let robotic marionettes execute entire "plays" by switching between different motions, as is shown in Figure 5. Moreover, assume that there are multiple robotic marionettes acting on the same stage, having to coordinate their motions with each other. In order to script motion programs for such scenarios, we modify the standard MDL to account for four important properties of our desired motion programs: *who* should act, *what* motion should they do, *where* should they operate, and *when* should the action occur, as discussed in [13].



(a) Puppet in initial configuration. (b) Puppet in wave motion. (c) Puppet starting a walk. (d) The final step in the walk mode.

Fig. 5 An image sequence of the puppet executing a *wave* followed by a *walk* mode.

We assume that the puppets are identified by an index, $i \in \mathcal{M}$, where $\mathcal{M} = \{1, \cdots, m\}$, and each puppet has the dynamics,

$$\dot{x}^i = f(x^i, u^i),\ x^i \in \mathbb{R}^n,\ u^i \in \mathbb{R}^p, \tag{12}$$

where we use the superscript i to denote agent-i.

We define the input to this model as one in a collection of possible feedback laws, i.e. $u^i = \kappa_j(x^i, t, \alpha_j)$, with κ_j, for some j, coming from a finite set of control laws $\mathcal{K} = \{\kappa_1, \cdots, \kappa_C\}$; additionally, α_j is an "energy"-scaling parameter that could affect speed, amplitude, or some other property of the control mode. When applying a controller of this form, we get the resulting closed-loop system dynamics $\dot{x}^i = f(x^i, t, \kappa_j(x^i, t, \alpha_j))$. And, by combining controllers from $\mathcal{K}$ with a time-driven interrupt, denoted τ, that dictates the time at which the control mode interrupts, resulting in controller-interrupt pairs of the form (κ, τ). However, to allow for the specification of programs involving multiple agents, we add in an element for agent identification, i, and a spatially defined location, r, where the agent performs its control κ. These locations in the environment come from a set $\mathcal{R} = \{r_1, \cdots, r_l\}$.

Now that we have modified MDL for composing multi-agent motion programs, we focus on developing a process for tweaking the timing and scaling parameters. For instance, an undesirable MDL mode would use a control law that potentially drives a system out of its intended operational region. It would be better to adjust the timing and energy scaling of the mode so that this behavior is prevented. We approach this problem using calculus of variations to derive optimality conditions that form the basis of an MDL compiler algorithm. This algorithm accepts a nominal motion program and outputs control code based on the system dynamics, under spatio-temporal constraints.

Say we are given a single-agent (agent-i) program with N modes over the time interval $[t_0, t_f]$, and we denote all switch time parameters as the vector $\bar{\tau}^i = [\tau_1^i \ \cdots \ \tau_{N-1}^i]$ and the scaling parameters as $\bar{\alpha}^i = [\alpha_1^i \ \cdots \ \alpha_N^i]$. Additionally, we denote the *nominal* switch times and energy parameters from our spatio-temporal MDL specification as $\hat{\tau}^i = [\hat{\tau}_1^i \ \cdots \ \hat{\tau}_{N-1}^i]$ and $\hat{\alpha}^i = [\hat{\alpha}_1^i \ \cdots \ \hat{\alpha}_N^i]$, respectively. Then the cost functional for optimizing this agent's program could take the form,

$$\min_{\bar{\tau}^i, \bar{\alpha}^i} J(\bar{\tau}^i, \bar{\alpha}^i) = \int_{t_0}^{t_f} L(x^i, t)dt + \sum_{j=1}^{N} \left(C_j(\alpha_j^i, \hat{\alpha}_j^i) + \Psi_j(x^i(\tau_j^i)) \right) + \sum_{k=1}^{N-1} \Delta_k(\tau_k^i, \hat{\tau}_k^i). \quad (13)$$

The interpretation here is that the agent has a trajectory cost, $L(x^i, t)$, associated with the execution of the motion program. Since scaling controller speed or amplitude requires more energy, we penalize the energy usage of each mode with the $C_j(\alpha_j^i, \hat{\alpha}_j^i)$ functions. In this general form, the energy penalty function could use the nominal values from the initial MDL specification, $\hat{\alpha}_j^i$. We also encode the spatial constraint for each mode through the spatial cost term, $\Psi_j(x^i(\tau_j^i))$, that penalizes the distance of the agent from the location of the specified region. Finally, to prevent large deviations of a particular switch-time τ_j^i, we add the temporal cost function $\Delta_k(\tau_k^i, \hat{\tau}_k^i)$, which uses the nominal switch-times, $\hat{\tau}_k^i$.

Assume that we construct an MDLp motion program of length N, which induces the system dynamics:

$$\dot{x}^i = \begin{cases} f(x^i, t, \kappa_1(x^i, t, \alpha_1)), \ t \in [\tau_0, \tau_1) \\ f(x^i, t, \kappa_2(x^i, t, \alpha_2)), \ t \in [\tau_1, \tau_2) \\ \vdots \\ f(x^i, t, \kappa_N(x^i, t, \alpha_N)), \ t \in [\tau_{N-1}, \tau_N] \end{cases} \quad (14)$$

where $\tau_0 := t_0$ and $\tau_N := t_f$. The following result gives us the optimality conditions necessary to "compile" this motion program. Note that this theorem was given (in a slightly different context) in [13], and is based on the timing control algorithms developed in [6].

Theorem 4 (First Order Necessary Optimality Conditions) *The first order necessary conditions for minimizing the cost functional* (13) *such that the system dynamics* (14) *are satisfied, are given by*

$$\begin{aligned} \frac{\partial J}{\partial \tau_k^i} &= \lambda^i(\tau_k^{i-})^T f_k(x(\tau_k^i), \alpha_k^i) - \lambda^i(\tau_k^{i+})^T f_{k+1}(x^i(\tau_k^i), \alpha_{k+1}^i) + \frac{\partial \Delta_k^i}{\partial \tau_k^i} \\ &\quad + \frac{\partial \Psi_k}{\partial x^i}^T f_{k+1}^i(x^i(\tau_k^i), \alpha_{k+1}^i) = 0, k = 1, \cdots, N-1 \quad (15) \\ \frac{\partial J}{\partial \alpha_k^i} &= \mu^i(\tau_{k-1}^{i+}) = 0, \ k = 1, \cdots, N \end{aligned}$$

where $f_k(x^i(t), \alpha_k^i)$ *denotes* $f(x^i, t, \kappa_k(x^i, t, \alpha_k^i))$ *and* τ_k^{i-} *and* τ_k^{i+} *are the left and right limits, respectively. Additionally, the discontinuous co-states* $\lambda^i(t) \in \mathbb{R}^n$, $\mu^i(t) \in \mathbb{R}$ *satisfy the co-state dynamics,*

$$\begin{aligned} \dot{\lambda}^i(t)^T &= -\frac{\partial J}{\partial x^i}^T - \lambda^i(t)^T \frac{\partial f_k}{\partial x^i} \\ \dot{\mu}^i(t) &= \lambda^i(t)^T \frac{\partial f_k}{\partial \alpha_k^i} \end{aligned}$$

with $t \in [\tau_{k-1}, \tau_k]$ *and boundary conditions,*

$$\begin{aligned}
\lambda^i(\tau_N^i) &= \frac{\partial \Psi_N}{\partial x^i}(x^i(\tau_N^i)) \\
\mu^i(\tau_N^i) &= \frac{\partial C_N}{\partial \alpha_N^i} \\
\lambda^i(\tau_k^{i-}) &= \lambda^i(\tau_k^{i+}) + \frac{\partial \Psi_k}{\partial x^i}(x^i(\tau_k^i)) \\
\mu^i(\tau_k^{i-}) &= \frac{\partial C_k}{\partial \alpha_k^i}
\end{aligned}$$

for $k = 1, \cdots, N-1$ *and where we let we let* $\tau_N = t_f$ *and* $\tau_0 = t_0$.

4.2 Simulation Results

To illustrate the operation of this hybrid optimal control-based CPS compiler, consider three robotic puppets. and assume that we have created three possible open loop controllers for them to execute, namely

$$\mathcal{K} = \{\kappa_1 = \texttt{waveLeftArm}, \kappa_2 = \texttt{walk}, \kappa_3 = \texttt{walkInCircles}\}.$$

For example, the `walk` mode applies alternating sinusoids of the form

$$u(t) = \frac{4}{\pi}\alpha\omega_{max}(\sin(2\pi f t) + (1/3)\sin(6\pi f t)),$$

where ω_{max} is a constant maximum rotational speed of the arm and leg lifting actuators and α is the scaling parameter for the control; see e.g. [12].

Using these controllers, we constructed the following puppetry play specified as a motion description language:

$$\begin{aligned}
&(p^1, \kappa_1(1.2), r_1, 2.5)(p^1, \kappa_2(1.3), r_1, 3)(p^1, \kappa_3(1), r_1, 4) \\
&(p^2, \kappa_1(1.2), r_3, 2.5)(p^2, \kappa_3(1.5), r_3, 2)(p^2, \kappa_2(1.3), r_3, 3) \\
&(p^3, \kappa_3(1), r_2, 2)(p^3, \kappa_2(1.5), r_2, 4).
\end{aligned}$$

The initial run of this play is illustrated by the *gray* lines and shapes in Figure 6. Note that puppets 1 and 2 (located in r_1 and r_3 in the figure) behave relatively well using their nominal plays. However, puppet 3 breaches the boundary between r_1 and r_2 while its nominal MDL string requires it to remain in r_2.

After running the MDL compiler on these strings, the improved runtime behavior is illustrated by the black lines and shapes in Figure 6. Puppet 3's trajectory is now within r_2, as prescribed in the original MDLp string. Also, all three puppets reduce their cost, as shown in Figure 7. Note that puppet 3 takes the longest, computing 100 iterations before minimizing its cost. This iteration count shows how bad the nominal program was at satisfying the cost functional (13). Additionally, our algorithm uses a conservative, fixed-step gradient descent to limit the amount of numerical error, which will slow down convergence as the derivatives (15) get closer to 0. If a dynamic step size were used (such as the Armijo step-size) then convergence would be faster. This work demonstrates that we can solve the problem of improving the multi-agent motion program given spatial costs. We now turn to the example of generating optimized control code under networked timing constraints.

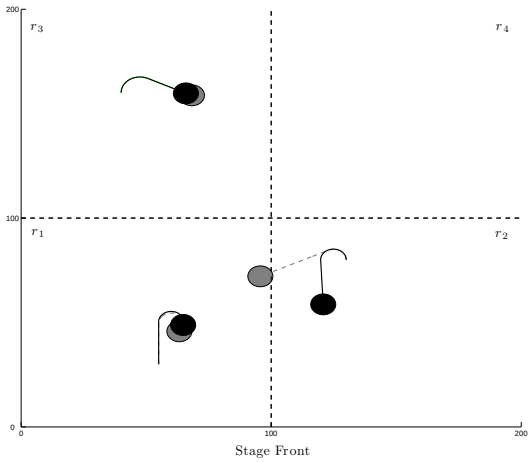


Fig. 6 Image of the puppet motions before (gray) and after (black) the MDL compilation process.

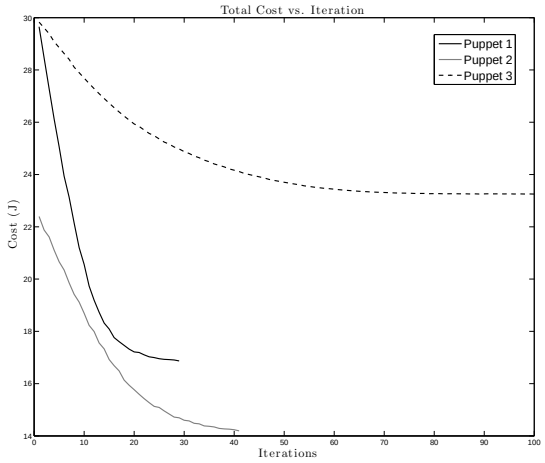


Fig. 7 This figure shows the costs as a function of the MDL compiler algorithm iteration when compiling a play for three puppets with *spatial* constraints. Puppet 1 completed in 29 iterations, Puppet 2 completed in 41 iterations, and Puppet 3 took 100 iterations.

5 Conclusions and Outlook

In this paper, we introduce the specification-execution gap for cyber-physical systems and show, through two distinct different classes of problems, that hybrid systems tools are natural selections for addressing this issue. The first problem we consider involves systems with saturation limits on the actuators and, as a result, we introduce a constraint-based compilation process that involves inserting new control modes into

the nominal string of modes that make up the high-level system specification. The second issue we consider is that of having specifications that do not take the actual system dynamics into account when specifying high-level control programs. Through optimal timing results for hybrid systems, we design a new type of CPS compiler that modifies the control laws and the interrupt conditions in order to make the resulting, new control string optimize a given cost functional.

The two particular sources of the specification-execution gap under investigation here are (1) constraint-based where the physical constraints of the system make the high-level program infeasible, and (2) specification-based where the specifications can be changed slightly in order to improve the performance of the system. This view of the need for compilers for CPS is quite general even though we, in this paper, only provided solutions to two rather particular instantiations. For a more general theory of control program compilation to be developed, additional and richer system and constraint classes must be addressed, including nonlinear and networked systems under actuation, power, communications, computation, and sensing constraints. This is, however, an undertaking that is both massive and important and it will no doubt combine both model-based, simulation-based, and constraint-based approaches. We hope, however, that this paper will serve as a starting point for this general line of investigation.

References

1. B. Archer, S. Sastry, A. Rowe, and R. Rajkumar. Profiling Primitives of Networked Embedded Automation, *Proceedings of the Fifth Annual IEEE Conference on Automation Science and Engineering (CASE 2009)*, August 2009.
2. 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. 116, Jan. 2006.
3. R.W. Brockett. On the Computer Control of Movement. In the *Proceedings of the 1988 IEEE Conference on Robotics and Automation*, pp. 534–540, New York, April 1988.
4. R.R. Burridge and A.A. Rizzi and D.E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research*, Vol. 8, No. 6, pp. 534-555, June, 1999.
5. Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, A Formal Approach to Deployment of Robotic Teams in an Urban-Like Environment, *10th International Symposium on Distributed Autonomous Robotics Systems (DARS)*, Lausanne, Switzerland, 2010
6. M. Egerstedt, Y. Wardi, and H. Axelsson. Transition-Time Optimization for Switched Systems. *IEEE Transactions on Automatic Control*, Vol. 51, No. 1, pp. 110-115, Jan. 2006.
7. 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. 10771091, Dec. 2005.
8. E. Klavins, A Formal Model of a Multi-Robot Control and Communication Task, *42nd IEEE Conference on Decision and Control.* Dec. 2003.
9. V. Manikonda, P. S. Krishnaprasad, and J. Hendler. Languages, behaviors, hybrid architectures and motion control. In J.C. Willems and J. Baillieul (Eds.) *Mathematical Control Theory*, Springer-Verlag, 1998.
10. P. Martin and M. Egerstedt. Expanding Motion Programs Under Input Constraints. In *Proceedings of American Control Conference*, Baltimore, Maryland, June, 2010.
11. On the Specification and Execution of Motion Programs for Networked Systems. *19th International Symposium on Mathematical Theory of Networks and Systems*, July 2010.
12. P. Martin and M. Egerstedt. Optimization of Multi-Agent Motion Programs with Applications to Robotic Marionettes. *Hybrid Systems: Computation and Control*, Springer-Verlag, 2009.
13. P. Martin and M. Egerstedt. Timing control of switched systems with applications to robotic marionettes. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, Vol. 20, No. 2, 2010.

14. J. Stankovic and I. Lee and A. Mok and R. Rajkumar. Opportunities and Obligations for Physical Computing Systems. *Computer*. Vol. 38, No. 11, pp. 23-31, November, 2005.
15. P. Twu, P. Martin, and M. Egerstedt. Graph Process Specifications for Hybrid Networked Systems. *International Workshop on Discrete Event Systems (WODES)*, Berlin, Germany, Aug. 2010.