# Automatic Synthesis of Control Alphabet Policies

Anastasia Mavrommati and Todd D. Murphey

*Abstract*— **This paper presents a method for synthesis of control alphabet policies, given continuum descriptions of physical systems and tasks. First, we describe a model predictive control scheme, called switched sequential action control (sSAC), that generates global state-feedback control policies with low computational cost, given a control alphabet. During synthesis, sSAC alphabet policies are directly encoded into finite state machines using a cell subdivision approach. As opposed to existing automata synthesis methods, controller synthesis is based entirely on the original nonlinear system dynamics and thus does not rely on but rather results in a lower-complexity symbolic representation. The method is validated for the cart-pendulum inversion problem and the double-tank system. The approach presents an opportunity for real-time task-oriented control of complex robotic platforms using exclusively sensor data with no online computation involved.**

## I. INTRODUCTION

This paper constructs control alphabet policies (CAP) that achieve desired performance objectives, given a (nonlinear) system and finite set of constant control symbols (denoted as the control alphabet). A CAP can be understood as a mapping that uniquely assigns a control symbol from the alphabet $U$ to a state space partition in a set $L$ (see Definition 2). Symbolic control has been popular in the areas of robot control and motion planning [1] as a means to provide solutions for control on embedded systems with limited computational power [2]. Common symbolic control approaches include linear temporal logic (LTL) [3] and motion description languages (MDLs) [4]. Here, as in MDLs, we synthesize symbolic policies considering discretization at the controls level, i.e., symbols denote control modes forming a control alphabet. For example, in a helicopter-like vehicle, tasks like "land", "ascend", and "hover" might correspond to alphabet policies composed of constant control modes.

Our objective is to synthesize simple control alphabet policies for otherwise complex dynamic tasks, that can be stored in finite state machines and realized with digital computer tools (e.g. hardware accelerators [5]) inexpensively, i.e. without requiring online control calculation. The solution to the problem of CAP synthesis for a wide range of systems and control objectives will benefit automation systems in terms of computing power allocation and compactness by boosting their multi-tasking capacity (power allocation) and promoting miniaturization (compactness), in the fields of aviation [6], manufacturing [7], and robotic locomotion [8] among others.

The proposed process of CAP synthesis extracts state space partitions $L$ to generate compact state policies described as a mapping $L \to U$ that assigns one control symbol from $U$ to each state space partition in $L$ instead of each state in $\mathcal{X}$. With this partitioning, control alphabet policies are structured as finite state machines during the synthesis process. Figure 1 shows how our approach (in red) compares to common methodologies for controller automata synthesis (in blue). The latter [2], [9]–[11] generate lower-complexity symbolic representations of continuous systems (i.e. bisimulation [11] or symbolic model [2]) and compute the policy on the system abstraction. On the contrary, this paper proposes that CAP synthesis (illustrated in blue) is entirely based on the original continuous nonlinear system dynamics instead of their lower-complexity abstractions. As a result, our objective is to provide solutions even for cases where a discrete system approximation is hard or impossible to obtain to aid in the controller synthesis. Subsequently, state-space abstractions are extracted based on the numerical global control policy. However, note that these state partitions do not aim to be a bisimulation of the actual system (as is formally defined in [11]) but a method for inexpensive representation of state-feedback controls and fast policy execution.

In order to achieve this alternative scheme, we formulate a method for state-feedback global control with a finite number of control symbols[1] that is computationally inexpensive and can be encoded in finite states machines. Common numerical approaches in hybrid control [12]–[14] output open-loop time-dependent control trajectories that do not favor the synthesis of global state-feedback control alphabet policies[2] while they exhibit high execution times. To overcome these issues, we propose a numerical algorithm for global symbolic control which we call switched sequential action control (sSAC). The algorithm is based on sequential action control (SAC), a recent model-based optimal control scheme described in detail in [15]–[18] and relies on hybrid systems theory to select the next symbol that optimally *improves* the task objective instead of optimizing it. While SAC outputs continuous control signals (unless saturated), sSAC outputs controls that switch among a finite number of control modes i.e. symbols.

Anastasia Mavrommati and Todd D. Murphey are with the Department of Mechanical Engineering, Northwestern University, 2145 Sheridan Road Evanston, IL 60208, USA Email: stacymav@u.northwestern.edu; t-murphey@northwestern.edu

This material is based upon work supported by the National Science Foundation under awards CMMI-1200321 and IIS-1426961. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

[1]This type of control is also known as mode scheduling where control symbols denote control modes.

[2]In other words, common mode scheduling algorithms do not naturally assign a symbol $u$ at any state $x$.

The final step of the proposed synthesis process generates state-space partitions based on an sSAC control policy, using a cell subdivision approach, typically used for computation of invariant sets [19]. As opposed to [2], the approach employs a multi-resolution grid, so that the distribution of state space partitions is non-uniform. As a result, the number of discrete state partitions doesn't grow exponentially with respect to the dimension of the state space. Furthermore, since the non-uniform partitions are hyper-rectangles that agree with the control policy (see Definition 2 and 3), the guard equations of the finite state machines can be directly represented as nested state inequalities for fast controller execution.

For method validation, we construct control alphabet policies for cart-pendulum inversion, and double-tank fluid levels control. When used for cart-pendulum inversion, we show that this automated numerical process—with sSAC and two symbols—generates structurally identical results to the bang-bang analytical control law published in [20]. The last example illustrates CAP synthesis in systems with hybrid dynamics. Finally, we discuss alternative uses of the CAP policies as embedded "background" controllers around which online controllers (e.g. sSAC or SAC) work to achieve high-level objectives.

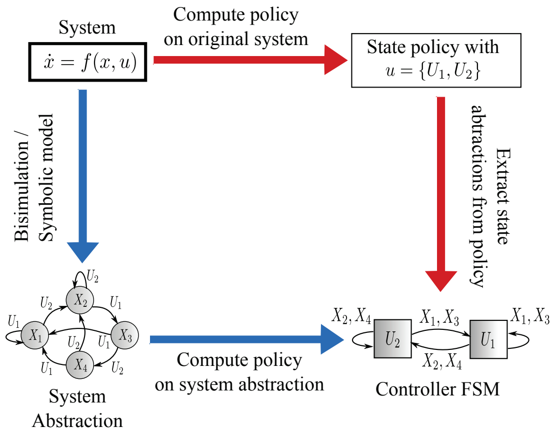
This paper is structured as follows: Section II introduces the problem, and is followed by Section III where switched sequential action control (sSAC) is presented. Section IV extracts state partitions from the policy using cell subdivision. In Sections V, VI, we synthesize policies for the cart-pendulum system and the double-tank system. Finally, a discussion on algorithm benefits, possible applications and future work is provided in Section VII.

## II. Problem formulation

We consider continuous-time nonlinear systems with $n$ states $x : \mathbb{R} \to \mathcal{X} \subseteq \mathbb{R}^n$ and $m$ inputs $u : \mathbb{R} \to U \subset \mathbb{R}^m$ following equations of the general form

$$\dot{x} = f(x, u). \tag{1}$$

At any time $t$, input $u(t)$ can be one of the $N$ constant-value control vectors from the set $U = \{u_1, ..., u_N\} \subset \mathbb{R}^m$. The state



System
$\dot{x} = f(x,u)$
Compute policy on original system
State policy with $u = \{U_1, U_2\}$
Bisimulation / Symbolic model
Extract state abtractions from policy
$U_1$
$U_2$
$X_1$
$X_2$
$X_3$
$X_4$
System Abstraction
Compute policy on system abstraction
$X_2, X_4$
$X_1, X_3$
Controller FSM

Fig. 1. A diagram showing how our approach (in red) compares to common automata synthesis methodologies [2], [9]–[11] (in blue).

is sometimes denoted as $t \mapsto x(t; t_0, x_0, u)$ when we want to make explicit the dependence on the initial time, initial state, and corresponding control signal.

*Assumption 1:* The elements of dynamics vector (1) are real, bounded, continuously differentiable in $x$, and continuous in $t$ and $u$.

*Definition 1:* A *symbol* is a constant-value control vector $u_i \in \mathbb{R}^m$ that belongs in the system's *alphabet*, i.e. control set $U$.

Our objective is to formulate a state-feedback control alphabet policy that satisfies a performance objective with low to no online computational cost. In contrast to e.g. [9], we do not assume any partition of the state space, so that controls are calculated for the original continuous system (1). However, during the synthesis process we will group sets of states to generate a multi-resolution partition $L$ of a compact set $\Omega$ in the state space $\mathcal{X}$ consisting of finitely many connected and disjoint subsets $L_i$ with the properties

$$\bigcup_{L_i \in L} L_i = \Omega \text{ and } L_i \cap L_j = \emptyset \;\; \forall L_i, L_j \in L, \;\; i \neq j. \tag{2}$$

Partition $L$ will be generated to intrinsically satisfy a numerical optimal control scheme, called switched sequential action control (sSAC), presented in the next section.

## III. Switched sequential action control

In this section, we present switched sequential action control (sSAC), a variation of sequential action control (SAC) in [15], that performs global closed-loop symbolic control using the symbols in $U$. The algorithm follows a receding-horizon approach; controls are obtained by repeatedly solving online an open-loop symbolic control problem $\mathcal{P}$ every $t_s$ seconds (with sampling frequency $\frac{1}{t_s}$), every time using the current measure of the system state $x_{curr}$. However, it differs from common receding-horizon schemes in two major points: a) $\mathcal{P}$ does not search for a control trajectory over the full time horizon $T$ but rather selects a single symbolic control action $u_{k^*} \in U$ to be applied for a short amount of time $\lambda$, and b) open-loop solution optimally improves the performance objective (3) instead of optimizing it. $\mathcal{P}$ improves general tracking objectives of the form

$$J(x(\cdot)) = \int_{t_0}^{t_0+T} l(x(t))\, dt + \bar{m}(x(t_0 + T)) \,, \tag{3}$$

with incremental cost $l(x(t))$, terminal cost $\bar{m}(x(t_0+T))$, initial time $t_0$ and time horizon $T$. The open-loop problem $\mathcal{P}$ is

$$\begin{aligned}
&\mathcal{P}(t_0, x_0, T, J) : \\
&\text{Find } k^* \in \{1, 2, ..., N\} \text{ and } \tau, \lambda \in \mathbb{R} \text{ such that} \\
&\qquad J(x(t; t_0, x_0, u_{sSAC})) < J(x(t; t_0, x_0, u_{default})) \\
&\qquad \text{with } u_{sSAC}(t) = \begin{cases} u_{k^*} & \tau \leq t \leq \tau + \lambda \\ u_{default} & \text{else} \end{cases} \\
&\qquad \text{subject to (1) with } t \in [t_0, t_0 + T] \text{ and } x(t_0) = x_0.
\end{aligned} \tag{4}$$

The term $u_{default}$ refers to a default (nominal) control symbol. It is often $u_{default} = \mathbf{0}$ so that problem $\mathcal{P}$ outputs the optimal symbolic action relative to doing nothing (allowing the system to drift for a horizon into the future). Alternatively, $u_{default}$ may be an optimized feedforward controller providing a nominal trajectory around which sSAC would provide feedback.

The solution $u_{sSAC}(t)$ of problem $\mathcal{P}$ generates a switch of duration $\lambda$ in the dynamics (1) from $f(x, u_{default})$ to $f(x, u_{k^*})$. The triplet $(k^*, \tau, \lambda)$—i.e. a single symbol $u_{k^*}$, $k^* \in \{1, 2, ..., N\}$ along with its associated application time $\tau$ and duration $\lambda$—defines a symbolic sSAC control *action*. As the receding horizon strategy progresses, $\mathcal{P}(t_0, x_0, T, J)$ is solved for the current time $t_0$ using the measured state $x_0$, and the output control $u_{sSAC}(t)$ is applied for $t_s$ seconds with $0 < t_s \leq T$. The process is then repeated at the next sampling instance, i.e. $t_0 \leftarrow t_0 + t_s$. This closed-loop receding horizon strategy, illustrated in Fig. 2, results in a sequence of symbolic actions, forming a piecewise constant control signal $\overline{u}_{cl}(t)$ with state response $\overline{x}_{cl}(t)$. With regard to CAP synthesis, note that in each cycle iteration and with $u_{default}$ determined, sSAC only takes as input the current state $x_0$ and outputs a single control symbol $u_{k^*}$ or $u_{default}$ to be applied for a finite duration at $t_0$. We take advantage of this natural state-dependence of sSAC controls in order to achieve synthesis of control policies in Section IV.

### A. *Solving open-loop problem $\mathcal{P}$*

To make explicit the dependence on action duration $\lambda$, application time $\tau$ and symbol $k^*$, we write inputs $u : \mathbb{R} \times \mathbb{R}^+ \times \mathbb{R} \times \{1, 2, ..., N\} \rightarrow U$ of the form of $u_{sSAC}(t)$ in (4) as

$$u(t; \lambda, \tau, k^*) = \begin{cases} u_{k^*} & \tau \leq t \leq \tau + \lambda \\ u_{default} & \text{else.} \end{cases}$$

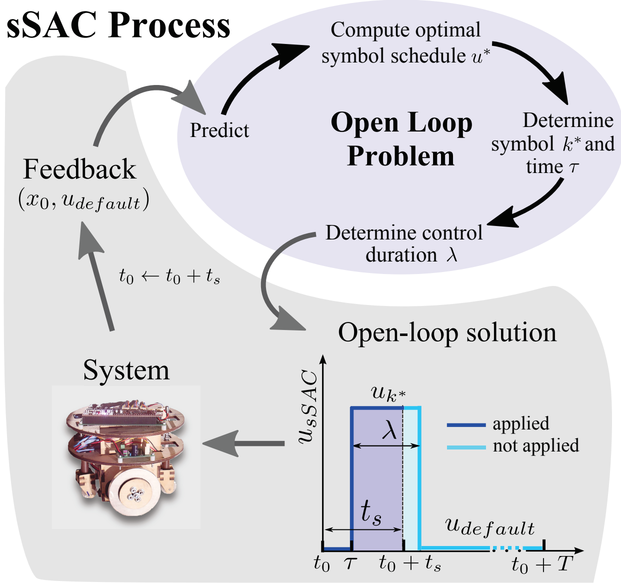


Fig. 2. The sSAC process takes as input the current state $x_0$ and repeatedly solves an open-loop problem $\mathcal{P}$ to synthesize piecewise constant control trajectories using the symbols in $U$.

When $\lambda = 0$, it is $u(t; 0, \cdot, \cdot) \equiv u_{default}$, i.e. no action is applied. Accordingly, we define $\bar{J}(\lambda, \tau, k^*) := J(x(t; t_0, x_0, u(t; \lambda, \tau, k^*)))$ so that the performance cost depends directly on the application parameters of a sSAC action. Using this notation, the open-loop problem $\mathcal{P}$ searches for the triplet $(k^*, \tau, \lambda)$ such that $\bar{J}(\lambda, \tau, k^*) < \bar{J}(0, \cdot, \cdot)$. There exists an open, non-zero neighborhood, $V = \mathcal{N}(\lambda \rightarrow 0)$, where the change in cost $\Delta J := \bar{J}(\lambda, \tau, k^*) - \bar{J}(0, \cdot, \cdot)$ is locally modeled by Taylor expansion as

$$\Delta J \approx \frac{d\bar{J}(\cdot, \tau, k^*)}{d\lambda^+} \lambda \tag{5}$$

for finite durations $\lambda \in V$. The quantity $\frac{d\bar{J}(\cdot,\tau,k^*)}{d\lambda^+}$—called mode insertion gradient and written $\frac{dJ}{d\lambda^+}\Big|_{\tau,k^*}$ for brevity—measures the first-order sensitivity of cost function (3) to application of symbol $u_{k^*}$ for infinitesimal duration $\lambda \rightarrow 0^+$ at time $\tau$. It is calculated as ( [21], [22])

$$\frac{dJ}{d\lambda^+}\Big|_{t,k} = \rho(t)^T (f(x(t), u_k) - f(x(t), u_{default})) \quad \forall t \in [t_0, t_0 + T]. \tag{6}$$

The adjoint variable $\rho : \mathbb{R} \rightarrow \mathbb{R}^n$ provides the sensitivity of (3) to state variations along a predicted trajectory $x(t)$ $\forall$ $t \in [t_0, t_0 + T]$. The adjoint satisfies[3]

$$\begin{aligned} \dot{\rho} &= -D_x l(x)^T - D_x f(x, u_{default})^T \rho \\ \text{subject to } & \rho(t_0 + T) = D_x \bar{m}(x(t_0 + T))^T. \end{aligned} \tag{7}$$

Expression (5) indicates that the difference $\Delta J$ depends on the value of the mode insertion gradient $\frac{d\bar{J}(\cdot,\tau,k^*)}{d\lambda^+}$ in (6) and is parameterized by the application time $\tau$ and duration $\lambda$. This relationship allows us to solve problem $\mathcal{P}$ by following the following four steps.

*1) Predict:* The sSAC process begins by predicting the evolution of a system model from current state feedback. In this step, the algorithm simulates system (1) from the current state $x_0$ and time $t_0$, for the finite horizon $[t_0,\ t_0 + T]$, under the default (nominal) control mode $u_{default}$. The prediction phase completes upon simulation of the adjoint system (7).

*2) Compute optimal symbol schedule $u^*$:* In this step, sSAC computes a schedule, $u^* : \{t \,|\, t \in [t_0, t_0 + T]\} \rightarrow U$, corresponding to the symbols that would optimally improve performance if applied for some duration at an arbitrary time $t \in [t_0, t_0 + T]$. To achieve optimal cost improvement, i.e. $\Delta J < 0$ in (5) or equivalently $J(x(t; t_0, x_0, u_{sSAC})) < J(x(t; t_0, x_0, u_{default}))$ in $\mathcal{P}$, the schedule selects at each time $t$ the symbol number $k$ that drives (6) the closest to a specified negative value, $\alpha_d \in \mathbb{R}^-$. Therefore, based on the simulation of (1) and (7) completed in the prediction step (Section III-A.1), the control schedule is calculated as

$$u^*(t) = \underset{u_k, k=1,...,N}{\text{argmin}} \left\{ \left[ \frac{dJ}{d\lambda^+}\Big|_{t,k} - \alpha_d \right]^2 + \|u_k\|^2_R \right\}, \quad t \in [t_0, t_0 + T]. \tag{8}$$

The matrix $R > \mathbf{0} \in \mathbb{R}^{m \times m}$ provides an optional[4] metric on control effort. Parameter $\alpha_d$ determines how smooth or aggressive the sSAC response will be, which is particularly

[3] $D_x f(\cdot)$ denotes the partial derivative $\frac{\partial f(\cdot)}{\partial x}$.

[4] The weight on $u_i$ is optional because there is only a finite number of control symbols.

useful when we have a large number of symbols and the controller is used in human-in-the-loop applications (e.g. see [16]). In any other situation, $\alpha_d$ can be eliminated with no change on the controller's effect.

*3) Determine application time $\tau$ and symbol $k^*$:* As mentioned before, the quantity $\frac{dJ}{d\lambda}$ parameterizes an action by its application time $\tau$. As a result, sSAC optimizes a decision variable not normally included in control calculations – the choice of *when* to act. In particular, sSAC searches $u^*$ for a time that optimizes the trade-off between the cost of waiting and the efficacy of control at that time. For more details, see [15]. Note that this step is *optional*, and is mostly employed when $u_{default} = \mathbf{0}$, i.e. when "doing nothing" is an option. In any other case, the application time is the "current" time $t_0$. Once $\tau$ is determined, the symbolic output of $\mathcal{P}$ in (4) is immediately specified as $k^* \in \{1, ..., N\}$ such that $u^*(\tau) = u_{k^*}$. In words, this step extracts from schedule $u^*(t)$ the single symbol $u_{k*}$ that corresponds to time $\tau$.

*4) Determine control duration $\lambda$:* The final step in synthesizing a sSAC action is to determine how long to act (select control duration $\lambda$). For this purpose, we use a line search with a simple descent condition to select a $\lambda$ that provides a minimum acceptable change in cost (3). In short, starting with a (short) initial duration, $\lambda = \lambda_0$, the effect of the control action is simulated from (1) and (3). If the simulated action improves cost (3) by a desired amount, the duration is selected. Otherwise, the duration is reduced and the process repeated. For a detailed description of this step, see [15].

After computing the duration, $\lambda$, the sSAC action is fully specified (it has a value, an application time and a duration) and the solution to open-loop problem $\mathcal{P}$ is complete.

## IV. Control alphabet policies

The objective of this paper is to generate control policies for inexpensive symbolic control. The resulting control laws are essentially hybrid automata, known to describe (discrete) switching conditions across a finite number of (continuous) dynamic modes [23]. Here, to stress the importance of the control alphabet policies (CAP) as standalone symbolic controllers relying on sensor data only, we use the following definition.

*Definition 2:* A *control alphabet policy* (CAP) is a 3-tuple $\langle U, L, \mathcal{T} \rangle$ where[5]:

— $U$ is the alphabet i.e a set of $N$ symbols;

— $L$ is a finite set of state space partitions that satisfy the sSAC control policy[6], i.e. $L = \{L_i \subset \mathcal{X},\ i = \{1, 2, ...\} : u_{sSAC}(0)\big|_{\mathcal{P}(\cdot, x_m, \cdot, \cdot)} = u_{sSAC}(0)\big|_{\mathcal{P}(\cdot, x_n, \cdot, \cdot)}\ \ \forall x_m, x_n \in L_i\}$;

— $\mathcal{T} : U \times L \rightarrow U$ is a (deterministic) transition function encoding a state-feedback control policy. This can be illustrated as a finite directed multigraph $(U, L)$, with elements in $U$ being the vertices and elements in $L$ the edges.

In this section, we describe a cell subdivision method that utilizes a non-uniform grid to extract state abstractions based on the symbolic sSAC policy[7].

Note here that sSAC sampling frequency $\frac{1}{t_s}$ in Fig. 2 has no practical meaning in the realization of the controller as a finite state machine (as in Definition 2). One way to think about it, is that the policy aims to capture the performance of sSAC when $t_s \rightarrow 0$. In practice during CAP application, sensor feedback rate is the primary limiting factor that determines frequency of state updates—not to be confused with sSAC frequency $\frac{1}{t_s}$.

**Algorithm 1** Compute Control Alphabet Policy

Initialize layer $k = 0$, desired maximum level $k_{max}$, time horizon $T$, number of test points $p \in \mathbb{N}$, set of state space partitions $L = \emptyset$, and set of compact sets to be divided $\Sigma^{(0)} = \{\Omega\}$ with $\Omega \subseteq \mathcal{X} \subseteq \mathbb{R}^n$ and $\Sigma^{(k)} = \emptyset\ \forall\ k > 0$.

```
1) For all compact sets $\Sigma^{(k,j)} \subset \mathbb{R}^n$, $j = 1, ...$ in $\Sigma^{(k)}$:
    a) Define a hyper-grid $\mathbf{C}^{(k,j)}$ on $\Sigma^{(k,j)}$ with $P^{(k,j)} \in \mathbb{N}$ cells
       and initialize labels $\ell(C_i^{(k,j)}) = Null$, $i = 1, ..., P^{(k,j)}$
       (unlabeled grid cells).
    b) For every grid cell $C_i^{(k,j)} \subset \Omega$, $i = 1, ..., P^{(k,j)}$:
        • Select $p$ test points $x_*$ from the cell interior or boundary.
        • For each point $x_*^{(s)} \in \mathbb{R}^n$, $s = 1, ..., p$, solve
          $\mathcal{P}(0, x_*^{(s)}, T, J)$ in (4). Then, $u^{(s)} = u_{sSAC}(0) \in U$.
        • If $u^{(s^1)} = u^{(s^2)}\ \forall\ s^1, s^2 \in \{1, 2, ..., p\}$ or $k = k_{max}$,
              label grid cell $C_i^{(k,j)}$ such that $\ell(C_i^{(k,j)}) = mode(\{u^{(s)} \in U : s = 1, ..., p\})$;
              add $C_i^{(k,j)}$ to set $L$;
          else
              add $C_i^{(k,j)}$ to set $\Sigma^{(k+1)}$.
2) $k \leftarrow k + 1$
3) Repeat from (1) until $\Sigma^{(k)} = \emptyset$ and $\bigcup_{L_i \in L} L_i = \Omega$.
```

### A. State abstractions using cell subdivision

*Definition 3:* Consider a compact set $\Omega \subset \mathbb{R}^n$ to a be a subset on the state space $\mathcal{X} \subseteq R^n$. A hyper-grid $\mathbf{C}$ on $\Omega$ divides $\Omega$ in a family of equally-sized $n$-orthotopes or hyper-rectangles[8] $C_i$, $i = 1, ..., P$, $P \in \mathbb{N}$ called cells, such that $\bigcup_{C_i \in \mathbf{C}} C_i = \Omega$ and $C_i \cap C_j = \emptyset\ \ \forall C_i, C_j \in \mathbf{C},\ \ i \neq j$. Each hyper-grid $\mathbf{C}$ is fully defined by $n$ sets of values $\bar{x}_i = \{x_i^{min}, x_i^{min} + \delta_i, x_i^{min} + 2\delta_i, ..., x_i^{max}\}$ that in turn define the partitioning of each state with $\delta_i$ the size of the $i^{th}$ cell dimension, so that each state $i$ has $q_i = \frac{x_i^{max} - x_i^{min}}{\delta_i}$ partitions. It is then $P = \{q_i\}^n$. To each cell $C_i$, associate a label $\ell(C_i) \in \mathbb{R}^m$ which can take values in $U$ or be $Null$ (i.e. cell is unlabeled).

Using the above definition, the complete process is given in Algorithm 1. The algorithm[9] takes as input a subset of the state space $\Omega \subseteq \mathcal{X}$ and outputs a partition $L$ of $\Omega$, i.e. $\bigcup_{L_i \in L} L_i = \Omega$ and $L_i \cap L_j = \emptyset\ \ \forall L_i, L_j \in L,\ \ i \neq j$. It is structured in layers starting from layer 0, with each new cell subdivision corresponding to a new layer. Initially, a

[5]From automata literature, control alphabet policies are similar in structure to labeled transition systems (LTS).

[6]That is, each state space partition in $L$ maps uniquely to one control symbol in $U$, such that the performance objective is satisfied.

[7]Note that the approach of this section can be applied without modification using any other global policy that outputs state-dependent controls based on a finite control alphabet.

[8]A $n$-orthotope or hyper-rectangle is the generalization of a rectangle for $n$ dimensions.

[9]In Algorithm 1, step b, $mode(A)$ denotes the value that appears most often in the set $A$.

standard—coarse—grid is applied on a state-space subset (layer 0). For each cell in the grid, if the labeling criterion is not satisfied, the cell is further subdivided to a finer grid of cells (e.g. layer 1) and the process repeats with finer layers until all cells are labeled whereafter partition $L$ has been fully specified. The $p$ test points in Step b can be the vertices, middle point and edge middle points of the cell as well as random interior points drawn from a distribution. A lower limit at the number of test points is $p_{min} = 2^n + 1$, i.e. the vertices and center point of the $n$-orthotope. Selection of number of test points $p$ is determined by the trade-off between desired accuracy level i.e. resolution ($p \uparrow$) and computational cost i.e. sSAC runs ($p \downarrow$).

The algorithm terminates when a maximum layer $k = k_{max}$ is reached, wherein the set of compact sets to be divided (unlabeled cells) is empty i.e., $\Sigma^{(k)} = \emptyset$. The algorithm must be terminated when $k = k_{max} < \infty$, so that all cells lying on sSAC switching manifolds are labeled and the boundaries of state partitions are resolved. The selected maximum level $k_{max}$ determines the lowest possible cell size of the resulting multi-resolution grid, which denotes the policy precision. The following proposition provides a method for selecting $k_{max}$ when the desired precision is known.

*Proposition 1:* In Algorithm 1, let the initial 0-level hyper-grid $\mathbf{C}^{(0)}$ comprise $P^{(0)} = P_{init}$ cells with $q_i^{init} \geq 2$ partitions for each state $i$. In addition, let $\delta_i^{(k_{max})}$ denote the desired $i^{th}$ cell dimension at maximum level $k_{max}$ for all $i \in \{1, ..., n\}$. Then, the maximum layer $k_{max}$ can be calculated as

$$k_{max} = \max_{i \in \{1,...,n\}} \left\{ \min_j \left\{ k_j \in \mathbb{N} : k_j \geq \log_{q_i^{inter}} \frac{x_i^{max} - x_i^{min}}{q_i^{init} \delta_i^{(k_{max})}} \right\} \right\}. \quad (9)$$

*Proof:* At a specified level $k$, the size of the $i^{th}$ cell dimension is computed as $\delta_i = \frac{x_i^{max} - x_i^{min}}{q_i^{init} (q_i^{inter})^{k_{max}}}$ (this result follows directly from the practice of subdivision in Algorithm 1). Solving for $k$ gives us a lower limit for the level $k$ with respect to desired precision $\delta_i$ as in (9). Then for each state $i$, we select the minimum natural number $k_j$ that satisfies the lower limit. Finally, $k_{max}$ is the maximum $k_j$ across all states $i = 1, ..., n$. ■

The computational cost of the algorithm primarily depends on the number of required sSAC runs, i.e. how many times we need to solve the open loop problem $\mathcal{P}(\cdot,\cdot,\cdot,\cdot)$ in Step 1b of Algorithm 1. If a fine uniform grid was used for state space discretization, the number of sSAC runs would be equal to the total number of grid points. However, here we use a multi-resolution grid, that is constructed starting from a coarse uniform grid at level 0 with each cell further subdivided to smaller cells until labeled. This can significantly reduce the number of grid points that require sSAC calculation. The more cells are labeled at level $k < k_{max}$, the less sSAC runs are executed. In this case, the exact number of sSAC runs is not known a priori and depends on how many cells will be labeled at each level (according to the labeling criterion). As a reference point, the following proposition provides an upper limit for the number of executed sSAC runs that is only reached in the worst case scenario when all cells are labeled at the maximum level $k_{max}$.

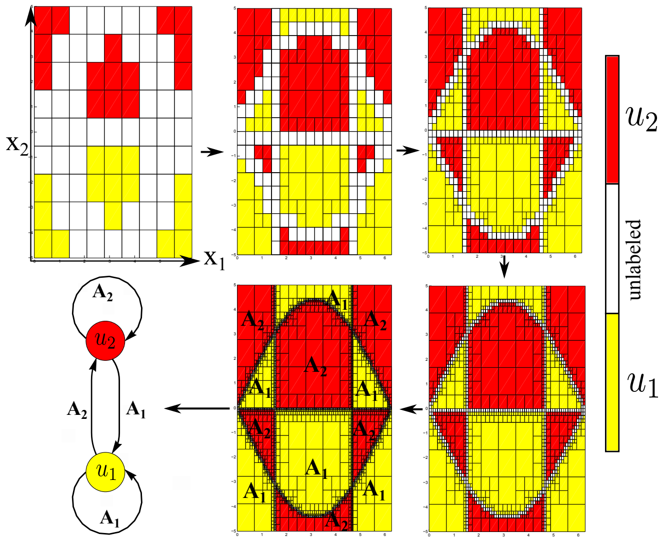


Fig. 3. Control alphabet policy for cart-pendulum inversion. Phase plane plots show consecutive layers of the cell subdivision strategy described in Algorithm 1 (layer 0 to layer 4) for synthesis of a two-symbol CAP with $U = \{u_1, u_2\} = \{5, -5\}$. The CAP finite state machine is also visualized. Vertices are the control symbols in $U$ and each edge label $A_i$ corresponds to the union of grid cells labeled with the symbol $u_i$.

*Proposition 2:* In Algorithm 1, let the initial 0-level hyper-grid $\mathbf{C}^{(0)}$ comprise $P^{(0)} = P_{init}$ cells with $q_i^{init} \geq 2$ partitions for each state $i$, and each lower-level hyper-grid $\mathbf{C}^{(k,j)}$ comprise $P^{(k,j)} = P_{inter}$ cells with $q_i^{inter} \geq 2$ partitions for each state, for all $j \in \mathbb{N}$ and $k > 0$. Then, if $p = 2^n + r$ test points (i.e. $2^n$ $n$-orthotope vertices and $r$ additional points from the cell interior) are selected for each cell $C_i^{(k,j)}$ $\forall\, k, j, i$, then the maximum possible number of sSAC runs in Algorithm 1 (i.e. worst-case scenario if all cells are labeled at level $k_{max}$) is

$$\text{runs} = \prod_{i=1}^{n} \left[ q_i^{init} (q_i^{inter})^{k_{max}} + 1 \right] + (1 + (P_{inter})^{k_{max}}) P_{init} r. \quad (10)$$

*Proof:* The first term computes the number of vertices of the hyper-grid $\mathbf{C}^{(k_{max})}$ with $q_i = q_i^{init}(q_i^{inter})^{k_{max}}$ partitions for each state $i$ at maximum level $k = k_{max}$. As $\mathbf{C}^{(k_{max})}$ consists of $P^{(k_{max})} = (1 + (P_{inter})^{k_{max}}) P_{init}$ cells, the second term quantifies the additional number of sSAC runs due to interior test points $r$. ■

So if the test points are $p = 2^n + 1$, (i.e. the vertices and center point of each $n$-orthotope cell), and the number of inner cells is $P_{inter} = 2^n$ with $q_i^{inter} = 2$ partitions for each state $i$ at each level $k$, the maximum number of sSAC runs is equal to $\prod_{i=1}^n \left[ q_i^{init} 2^{k_{max}} + 1 \right] + (1 + 2^{nk_{max}}) P_{init} r$. This number increases exponentially with both the number of levels $k_{max}$ and the number of dimensions $n$. However, since we are using a multiple-level non-uniform grid, the *actual number of sSAC runs* is expected to be significantly lower as more cells are labeled at levels $k < k_{max}$. Whether the methodology is scalable to higher dimensions largely depends on the sparsity of controls over the system's state space and differs among choice of systems and symbols.

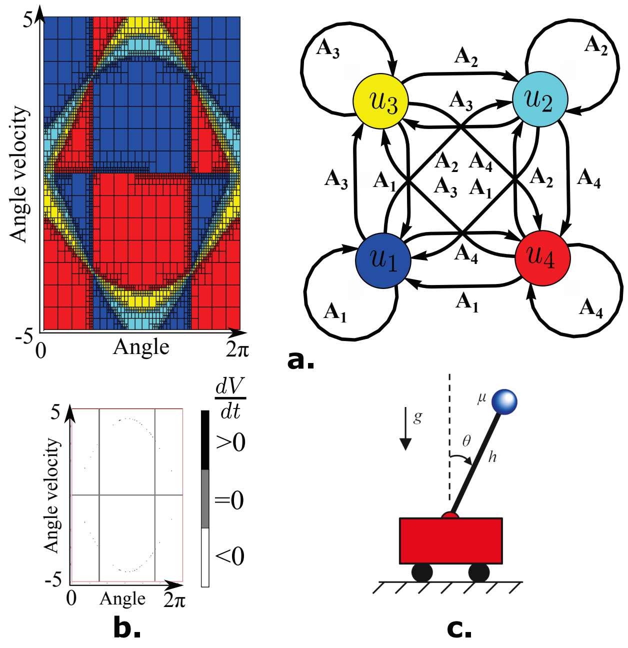


Fig. 4. Control alphabet policies for cart-pendulum inversion. (a) Phase plane plot and finite state machine showing the sSAC-generated four-symbol CAP with $U = \{u_1, u_2, u_3, u_4\} = \{-5, -2, 2, 5\}$. (b) Numerical evaluation of the Lyapunov derivative on the control policies. Derivative is negative (stable) except for the gray lines $\dot{\theta} = 0$ and $cos\theta = 0$, where it is zero. These lines correspond to system singularities and are not concerning with regard to stability, as it happens that the vector field always drives the system out of these regions (non-invariant sets). (c) The cart-pendulum system.

## V. Example: cart-pendulum inversion

This section demonstrates how to numerically synthesize control alphabet policies that globally lead to pendulum swing-up using full state feedback and a specified finite number of symbols. The reduced cart-pendulum system, shown in Fig. 4c, has $n = 2$ state variables, the angle between the vertical and the pendulum $\theta$ and the rate of change of the angle $\dot{\theta}$. Denoting by $h$ the pendulum length, $g$ the gravity acceleration and $\mu$ the mass, the system equations take the form in (1) with

$$f(x,u) = \begin{pmatrix} \dot{\theta} \\ \frac{g}{h} sin\theta + \frac{u}{h} cos\theta \end{pmatrix}, \quad x = [\theta, \dot{\theta}] \tag{11}$$

where $u \in \mathbb{R}$ is the acceleration of the cart (i.e. $m = 1$). The pendulum is inverted when $(\theta, \dot{\theta}) = (0, 0)$. The system parameters take values $h = 2$, $\mu = 1$ and $g = 9.81$. Using this model, we synthesize control alphabet policies for symbolic cart-pendulum inversion by tracking the energy of the pendulum at the upright position. The energy of the uncontrolled pendulum ($u = 0$) is $E(\theta, \dot{\theta}) = \frac{1}{2}\mu h^2 \dot{\theta}^2 + \mu g h(cos\theta - 1)$ so that $E_0 = 0$ at the upright unstable equilibrium. For sSAC controls computation, we use the cost function (3) with $t_0 = 0$,

$$l(x(t)) = 0 \quad \text{and} \quad \bar{m}(x(t_f)) = \frac{1}{2}(E(\theta(t_f), \dot{\theta}(t_f) - E_0)^2. \tag{12}$$

In addition, for the open-loop problem $\mathcal{P}$, we used the parameters: $T = 1.2$, $u_{default} = 0$, $\alpha_d = -5J$, $R = 0.3$.

Figure 3 and Fig. 4a illustrate the resulting control alphabet policies for energy tracking using two symbols $U = \{-5, 5\}$ and four symbols $U = \{-5, -2, 2, 5\}$, respectively. It is noteworthy that the two-symbol control policy is structurally identical to the analytical bang-bang solution provided in[10] [20], albeit a result of a completely automated numerical procedure. In addition, we now extend these analytical control laws and derive symbolic policies for a larger, arbitrary number of symbols.

The computation was done on the compact set $\Omega = [0, 2\pi] \times [-5, 5]$ with a starting grid (layer 0) consisting of $P^{(0)} = 81$ grid cells and $q_i^{(0)} = 9$ partitions in each state $i$. In finer layers, cells were subdivided in grids of $P^{(k,j)} = 4 \ \forall \ k > 0, j \in \mathbb{N}$ cells. With $p = 5$ test points per cell and maximum level $k_{max} = 4$, a total of 3428 sSAC runs for the two-symbol policy and 8981 runs for the four-symbol policy were performed, compared to 41842 runs that would be required if all cells were labeled at maximum level 4 (computed using the expression in (10)). The resulting CAP are verified for Lyapunov stability next.

*a) Lyapunov stability:* For stability verification of the control laws, we use the Lyapunov function from [20], $V = |E - E_0|$ with $\frac{dV}{dt} = sign(E)\dot{E}$ and $\dot{E} = \mu \cdot h \cdot u \cdot cos\theta \cdot \dot{\theta}$. We numerically verified the value of the Lyapunov derivative for all states $x \in [0, 2\pi][-5, 5]$ (using a state-space discretization of grid size 0.01), and for both control policies $u$ in Fig. 3 and Fig. 4a. The result (i.e. $sign(\frac{dV}{dt})\forall x$) is identical for both policies and is illustrated in Fig. 4b. The Lyapunov function decreases (i.e. $\frac{dV}{dt} < 0$) as long as $\dot{\theta} \neq 0$ and $cos\theta \neq 0$ (gray lines). Implementation-wise, these lines (where $\frac{dV}{dt} = 0$) are not concerning with regard to stability, as it happens that the vector field always drives the system out of these regions (non-invariant sets). Scattered black points ($\frac{dV}{dt} > 0$) on the switching manifold are due to numerical noise generated by the grid discretization and numerical integration in sSAC.

## VI. Example: Two-tank system

This section synthesizes control alphabet policies that track desired fluid levels at a double-tank system. Figure 5 shows the configuration of the system that consists of two tanks $T_1$ and $T_2$, with $T_1$ elevated at a height $h$ with respect to $T_2$. This tank configuration is a common laboratory setting and variants of it have been extensively used for evaluation of control methodologies [13], [24]. The inflow and outflow rates to the tanks are controlled by the valves $V_1$, $V_2$, and $V_3$, that can only be open with flow rate $V_i = 1$ or closed

[10]The bang-bang control law in [20] is $u = |u_i| sign((E - E_0)\dot{\theta} cos\theta)$.

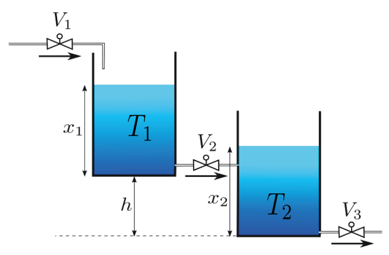


Fig. 5. The two-tank system configuration.

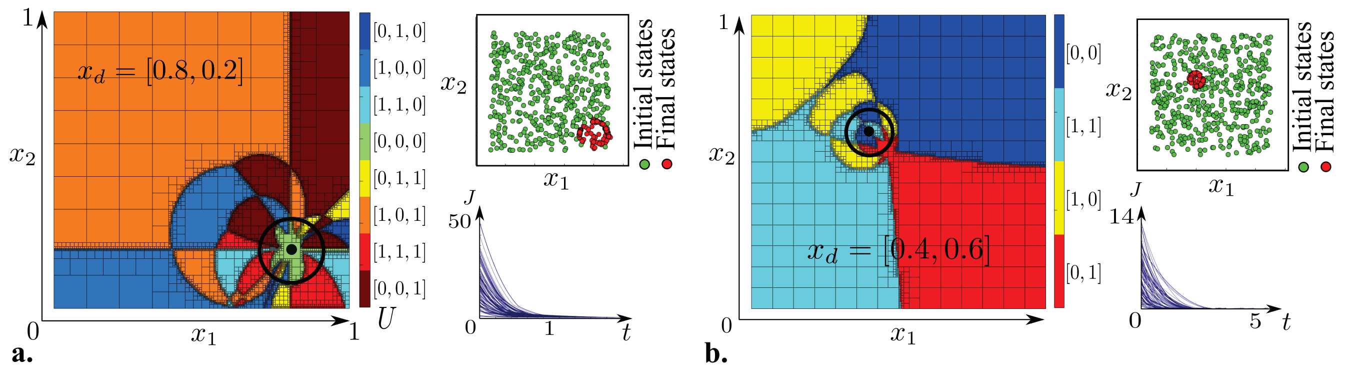


Fig. 6. Control alphabet policies for tracking desired fluid levels in a double-tank system. (a) Case A: Phase plane plot showing the CAP policy with $u = [V_1, V_2, V_3]$ and $N = 8$ symbols. Desired state is $x_d = [0.8, 0.2]$. Figures on the right show Monte Carlo results with tolerance $\epsilon = 0.1$ (depicted as a circle). A 100% rate of success was achieved. Blue trajectories at the bottom right figure show reduction of open-loop cost $J$ in (3) over time for a sample of 100 trials. Note that $J$ was only calculated for verification purposes and was not part of the control calculation. Top right figure shows a phase plane plot with the initial trial states in green and the final states in red for 500 trials. (b) Case B: Phase plane plot showing the CAP policy with $u = [V_1, V_3]$, $V_2 = 0.2$ and $N = 4$ symbols. Desired state is $x_d = [0.4, 0.6]$. Monte Carlo test was performed with tolerance $\epsilon = 0.05$. A 100% rate of success was achieved. In all trials, open-loop cost $J$ in (3) was decreased over time.

so that $V_i = 0$. The states $x_1$ and $x_2$ are the fluid levels of tanks $T_1$ and $T_2$ respectively. According to Toricelli's law, a simplified model of the system consists of the nonlinear vector field

$$f(x,u) = \begin{cases} \begin{bmatrix} V_1 - V_2\sqrt{x_1 - x_2 + h} \\ V_2\sqrt{x_1 - x_2 + h} - V_3\sqrt{x_2} \end{bmatrix}, & \text{if } x_2 > h \\ \begin{bmatrix} V_1 - V_2\sqrt{x_1} \\ V_2\sqrt{x_1} - V_3\sqrt{x_2} \end{bmatrix}, & \text{else.} \end{cases} \qquad (13)$$

There is a (continuous) switch at the dynamics when $\Phi = x_2 - h$ crosses zero. The height takes value $h = 0.5$. For sSAC controls computation, we use the cost function (3) with $t_0 = 0$,

$$l(x(t)) = \|x(t) - x_d\|^2_Q \quad \text{and} \quad \bar{m}(x(t_f)) = \|x(t_f) - x_d\|^2_P \qquad (14)$$

where $x_d$ is the desired state. The weight matrices are $Q = Diag(\{1,1\})$ and $P = Diag(\{100,100\})$. In addition, for the open-loop problem $\mathcal{P}$, we used the parameters: $T = 0.5$, $u_{default} = 0^{m\times 1}$, $\alpha_d = -5J$.

To explore the potential of CAP synthesis using sSAC, we synthesized policies for two different cases of control authority: A. with $u = [V_1, V_2, V_3]$, so that all valves are controlled and B. with $u = [V_1, V_3]$, so that only two valves are controlled and the middle valve has a constant flow rate $V_2 = 0.2$. The resulting policies and more details on each example case are given in Fig. 6b,c.

For both control scenarios, we ran Monte Carlo tests with 1000 trials simulating system dynamics (13) from random initial states in the set $[0.1, 0.9] \times [0.1, 0.9]$. During simulation, control $u$ is generated by the CAP at 1000 Hz using the current state $x_{curr}$ and the CAP transition function $\mathcal{T}$ (see Definition 2). A trial terminates when the system is sufficiently close to the desired state as specified by tolerance $\epsilon$ i.e. $|x_{curr} - x_d| < \epsilon$ (successful trial), or when a time limit is exceeded i.e. $t > t_{max}$ (failed trial). Specifying a tolerance is consistent with previous results about quantized systems [25], stating that one can only implement feedback strategies that bring closed-loop trajectories arbitrarily close to the desired state. For both cases A and B, the rate of success was 100% in 1000 trials.

Furthermore, notice that CAP performance in case B is improved compared to case A (a 100% rate of success was achieved with lower tolerance around the desired state), because in case B, CAP is capable of exploiting the system's free dynamics (in case B, the system exhibits no free dynamics since all valves are controlled). This example shows how policies based on sSAC are generated to intrinsically exploit rather than cancel complex dynamical behaviors.

Computation of both policies (Algorithm 1) was done on the compact set $\Omega = [0,1]\times[0,1]$ with a starting grid (layer 0) consisting of $P^{(0)} = 81$ grid cells and $q_i^{(0)} = 8$ partitions in each state $i$. In finer layers, cells were subdivided in grids of $P^{(k,j)} = 4 \ \forall\ k > 0,\ j \in \mathbb{N}$ cells. With $p = 5$ test points per cell and maximum level $k_{max} = 4$, a total of 7,621 sSAC runs for case A and 3,965 runs for case B were performed, compared to 41,842 runs that would be required if all cells were labeled at maximum level 4 (computed using the expression (10)). Interestingly, we repeated the calculation for $k_{max} = 5$, wherein a total of 17,125 sSAC runs for case A and 9,474 runs for case B were performed, compared to 166,546 runs that would be required if all cells were labeled at maximum level 5. Therefore, although the worst-case scenario sSAC calculations increased exponentially, the actual number of calculations only almost doubled compared to the case with $k_{max} = 4$.

## VII. Discussion and Future Work

In this paper, we introduced an approach for compact controller synthesis that combines elements from two common methodologies for optimal nonlinear control: a) generation of optimal state policies that utilize a state-space discretization (e.g. Markov Decision Processes [26]) and b) model predictive control algorithms that apply open-loop finite-horizon control in a receding-horizon format [27]. The first generates a mapping $\mathcal{X} \to U$ but suffers from the curse of dimensionality while the second requires knowledge of future state trajectories in a specified time window in order to generate an equivalent mapping. We addressed the first drawback by restricting control to a finite number of control symbols and extracting state-space partitions $L$ so that a mapping $L \to U$ is achieved. Secondly, we introduced sSAC, a model predictive control approach that naturally assigns a symbol $u$ at any state $x$ by outputting a single control action instead of a full control trajectory. Notice that although sSAC does not calculate the optimal control trajectory (i.e. the trajectory that minimizes cost (3)), it still calculates the control action that *optimally* improves the cost, in the sense that it optimizes the change in cost $\frac{dJ}{d\lambda^+}$ in (8).

Our next step is to verify scalability of the method to high-dimensional systems both in simulation and experiment. As mentioned previously in the paper, computational cost will largely depend on the sparsity of control symbols over the system's state space. One method to maximize sparsity is to utilize a control alphabet that best serves the purposes of the performance objective. Therefore, future work will focus on establishing scalability guarantees by optimally "extracting" control symbols based on the system dynamics and the task to be performed.

The two examples (on the cart-pendulum and two-tank systems) demonstrated the CAP synthesis process and showed that sSAC can be efficiently encoded in a finite state machine as a state control policy. One thing to notice is that in both cases, the resulting state space partitions are reasonably sparse (i.e. a large number of cells are labeled at the lower levels). This result suggests that we can employ this method to generate simple and compact controllers that perform complicated dynamic tasks using nonlinear models. The benefits of compactness and power allocation promote alternative uses of the CAP policies as embedded "background" controllers around which online controllers (e.g. sSAC or SAC) work to achieve high-level objectives. For example, a CAP finite state machine embedded in a robotic biped can be used to inexpensively coordinate walking, while high-level controllers complete more complex tasks (similar to the hypothesis that spinal cord circuitry coordinates locomotion in humans and other vertebrates [28]). This application will be further explored through experimentation.

## References

[1] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 61–70, 2007.

[2] G. Pola, A. Girard, and P. Tabuada, "Approximately bisimilar symbolic models for nonlinear control systems," *Automatica*, vol. 44, no. 10, pp. 2508 – 2516, 2008.

[3] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *IEEE Conference on Decision and Control*, 2009, pp. 5997–6004.

[4] P. Martin and M. B. Egerstedt, "Hybrid systems tools for compiling controllers for cyber-physical systems," *Discrete Event Dynamic Systems*, vol. 22, no. 1, pp. 101–119, 2012.

[5] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *ACM/IEEE 41st International Symposium on Computer Architecture*. IEEE, 2014, pp. 97–108.

[6] X. Deng, L. Schenato, and S. S. Sastry, "Flapping flight for biomimetic robotic insects: Part II-flight control design," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 789–803, 2006.

[7] Z. Sun and L. Li, "Opportunity estimation for real-time energy control of sustainable manufacturing systems," *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 1, pp. 38–44, 2013.

[8] H.-W. Park, A. Ramezani, and J. Grizzle, "A finite-state machine for accommodating unexpected large ground-height variations in bipedal robot walking," *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 331–345, 2013.

[9] M. Mazo and P. Tabuada, "Symbolic approximate time-optimal control," *Systems & Control Letters*, vol. 60, no. 4, pp. 256–263, 2011.

[10] M. Broucke, M. D. Di Benedetto, S. Di Gennaro, and A. Sangiovanni-Vincentelli, "Theory of optimal control using bisimulations," in *Hybrid Systems: Computation and Control*. Springer, 2000, pp. 89–102.

[11] A. Girard and G. Pappas, "Approximation metrics for discrete and continuous systems," *IEEE Transactions on Automatic Control*, vol. 52, no. 5, pp. 782–798, 2007.

[12] H. Gonzalez, R. Vasudevan, M. Kamgarpour, S. S. Sastry, R. Bajcsy, and C. Tomlin, "A numerical method for the optimal control of switched systems," in *IEEE Conference on Decision and Control*, 2010, pp. 7519–7526.

[13] Y. Wardi and M. Egerstedt, "Algorithm for optimal mode scheduling in switched systems," in *American Control Conference, 2012*, 2012, pp. 4546–4551.

[14] A. Mavrommati, J. Schultz, and T. D. Murphey, "Real-time dynamic mode scheduling using single-integration hybrid optimization," *IEEE Transactions on Automation Science and Engineering*, vol. PP, no. 99, pp. 1–14, 2016.

[15] A. Ansari and T. D. Murphey, "Sequential action control: closed-form optimal control for nonlinear systems," *IEEE Transactions on Robotics*, Accepted, 2016. [Online]. Available: http://nxr.northwestern.edu/publications/sequential-action-control-closed-form

[16] A. Mavrommati, A. Ansari, and T. D. Murphey, "Optimal control-on-request: An application in real-time assistive balance control," in *IEEE International Conference on Robotics and Automation*, 2015, pp. 5928–5934.

[17] A. Ansari, K. Flaßkamp, and T. D. Murphey, "Sequential action control for tracking of free invariant manifolds," in *Conference on Analysis and Design of Hybrid Systems*, 2015.

[18] E. Tzorakoleftherakis, A. Ansari, A. Wilson, J. Schultz, and T. D. Murphey, "Model-based reactive control for hybrid and high-dimensional robotic systems," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 431–438, 2016.

[19] M. Dellnitz, G. Froyland, and O. Junge, "The algorithms behind GAIO—Set oriented numerical methods for dynamical systems," in *Ergodic theory, analysis, and efficient simulation of dynamical systems*. Springer, 2001, pp. 145–174.

[20] K. J. Åström and K. Furuta, "Swinging up a pendulum by energy control," *Automatica*, vol. 36, no. 2, pp. 287–295, 2000.

[21] M. Egerstedt, Y. Wardi, and H. Axelsson, "Transition-time optimization for switched-mode dynamical systems," *IEEE Transactions on Automatic Control*, vol. 51, no. 1, pp. 110–115, 2006.

[22] M. Egerstedt, Y. Wardi, and H. Axelsson, "Optimal control of switching times in hybrid systems," in *IEEE International Conference on Methods and Models in Automation and Robotics*, 2003.

[23] J. Lygeros, K. H. Johansson, S. N. Simić, J. Zhang, and S. S. Sastry, "Dynamical properties of hybrid automata," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 2–17, 2003.

[24] O. Stursberg and S. Engell, "Optimal control of switched continuous systems using mixed-integer programming," in *15th IFAC World Congress of Automatic Control*, 2002, p. 100.

[25] D. F. Delchamps, "Stabilizing a linear system with quantized state feedback," *IEEE Transactions on Automatic Control*, vol. 35, no. 8, pp. 916–924, 1990.

[26] A. Gorodetsky, S. Karaman, and Y. Marzouk, "Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition," *Robotics: Science and Systems XI, Sapienza University of Rome, Italy*, 2015.

[27] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[28] A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008.