 DOI: 10.3217/978-3-85125-663-5-15

A Dynamical System for Governing Continuous, Sequential and Reactive Behaviors

Raphael Deimel

***Abstract*— In interaction with humans or movable objects, robots not only need to react to surprising information quickly, but they also need to synchronize their motions with the world, which can be done by introducing decision points (discrete state transitions), or by continuously adjusting the execution velocity. We present a novel dynamical system based on stable heteroclinic channel networks that can represent static, markovian states as well as continuous transitions between states in a compact and consistent state vector. This so-called phase-state machine can implement regular state machine semantics, but it additionally has the built-in capability to provide and adjust phases and blend consecutive movement primitives for smooth operation. In this paper, we investigate the dynamic properties, present examples for programming specific state machine semantics, and demonstrate the sequencing and mixing of continuous movement primitives.**

I. Introduction

Behaviors involving interaction (e.g. human-robot object handover [14], reactive manipulation strategies [3]) can often be described by directed graphs that sequence simpler behavior generation systems. The most prominent formalism used is the hybrid automaton [15], which is a state machine that activates and deactivates specific controllers and trajectories. For fluid interactions though, we often want behaviors that violate the strict temporal separation between motion generators. We want consecutive motions to blend into each other. We want to preserve estimates by control (e.g. the weight of a tool) across control switches. Or we want to mix motions to communicate uncertainty to an interaction partner. None of these behaviors are impossible to implement using hybrid automata, but require us to fragment and obfuscate the state graph by introducing many additional states and transitions to handle deviating or unclear situations. In addition, to achieve reactivity in real systems it is common for controllers to bypass the state machine abstraction all together by exchanging information with other controllers or perception directly.

We believe that these unwieldy solutions are the symptom of a limitation of the conventional discrete state machine, which cannot represent transitions with nonzero duration between its states. Introducing continuous, time-extended, and nonexclusive (non-Markovian) transitions in between (Markovian) states provides an elegant compromise between providing a notion of time, and independent, separate segments at the same time. The former enables synchronization and blending, while the latter divides complex behaviors in to independent segments that are easier to learn, optimize, and reason with. System evolution becomes continuous at any point in time, while conventional state graphs can be recovered by ignoring transitions.

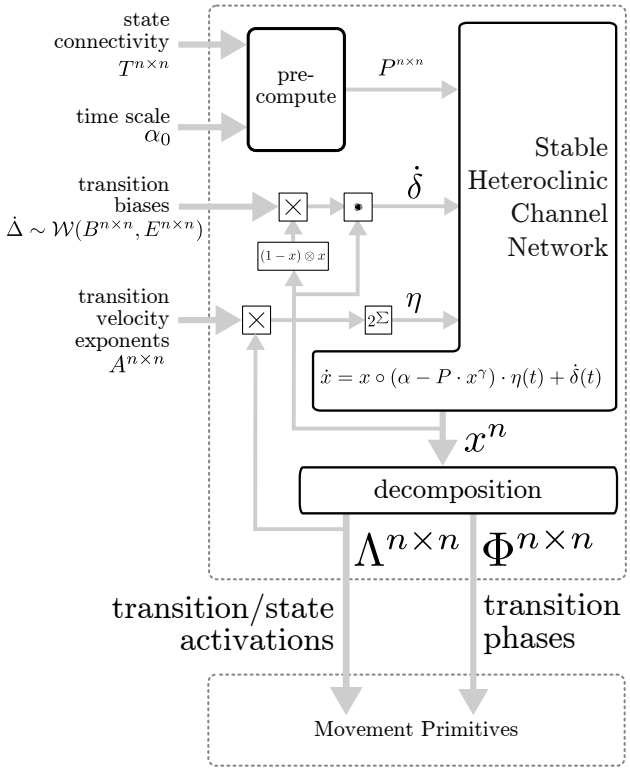


Fig. 1: By decomposing the system state into activation and phase values, dynamical systems with stable heteroclinic channels [4] can orchestrate phase-based movement primitives.

In order to provide continuous, time-extended, nonexclusive transitions, we could extend the hybrid automaton to provide explicit "transition states" in between control states. Though, one would need to add a state for each possible combination of transition and regular state, and for any combinations of transitions sharing the same predecessor state, as those could be active concurrently. In this paper, we investigate an entirely different approach: we implement a discrete state machine using a dynamical system instead of implementing a dynamical system using a discrete state machine. We achieve this by constructing a dynamical system whose attractor consists of a network of "stable heteroclinic channels" (SHC) [11], connected by saddle points which

The author is with the Control Systems Laboratory, Technische Universität Berlin, Germany

We gratefully acknowledge financial support for the project MTI-engAge (16SV7109) by BMBF .

can be interpreted as transitions and states respectively. In previous research, stable heteroclinic channels have been treated as discrete transitions [4], [11], the transitions are continuous in their nature and extended over time though. The SHC formalism provides a straightforward method to construct states (saddle points) and arbitrary transitions (channels) between those states. This paper extends the formalisms to assign an activation value for any possible state and any possible transition as well as to assign a phase variable for each transition. We further will demonstrate how to use those activation and phase values to adjust periods of each transition individually and how to integrate (perceptual) information continuously and context-dependent on the current state.

The activation and phase values can also be used directly to govern phase-parameterized movement primitives such as DMPs [13] or ProMPs [9], which we will demonstrate for a robotic arm.

II. Related Work

The problem domain addressed by the phase-state machine has traditionally been addressed with hybrid dynamical systems [15], and more specifically with hybrid automata. The main difference of phase-state machines w.r.t. hybrid automata is, that states are not necessarily digital, transitions are not discrete, and transitions with a common predecessor can stay in (unstable) superposition. Confusion may arise from the difference in semantics for state and state transition: In a hybrid automaton, trajectories or primitives happen during a *state*, while in the proposed phase-state machine trajectories happen during a *state transition*. Conversely, in hybrid automata synchronization barriers and delays are implemented by *state transitions* (guarded jumps), while in the the phase-state machine, it is implemented by dwelling in *states*.

The work on the phase-state machine builds on prior work that proposed a straightforward method to "program" states and transitions [11], [4]. Their systems have been used as central pattern generators [4], controllers were associated with states and not transitions, though, emulating a hybrid automaton. Phase-state machines are intended to govern a set of motion primitives such as ProMPs. [9], [6], DMPs [2] or others [8] to synthesize actual robot behavior. A recent publication on ProMPs proposes a library of actions and associated transition triggers to concatenate actions [5], albeit it focuses on segmenting and organizing primitives, and not on sequencing and synchronizing their execution.

III. Method

Fig. 1 shows all components of a phase-state machine. The core consists of a Lotka-Volterra-type differential equation, which evolves according to the equation:

$$\dot{x} = x \circ (\alpha - P \cdot x^{\gamma}) \cdot \eta(t) + \dot{\delta}(t) \tag{1}$$

where x is an n-dimensional state vector, vector α and matrix $P^{n \times n}$ are parameters defining the attractor landscape, and $\circ$ denotes element-wise multiplication (Hadamard product). The terms η and $\dot{\delta}$ are control inputs and used to modify the system behavior and will be explained in detail further down. Vector α is called the growth rate parameter, as it influences how fast the value of a state variable grows during a transition. P is a matrix that sets the excitation or inhibition between the state variables so that one saddle point on each coordinate axis is created to represent a discrete state. Further we assume $\dot{\delta} \sim \mathscr{W}(\mu, \sigma)$ to be generated by a Wiener process. The system is therefore numerically integrated using the Euler-Maruyama integration scheme:

$$x(t+\Delta t) = x(t) + E[\dot{x}(t)] \cdot \Delta t + \frac{\sqrt{\Delta t}}{\Delta t} \cdot \mathscr{N}(0, \sigma) \tag{2}$$

a) Inputs to influence system behavior: In order to influence the evolution of the dynamical system, we provide two inputs to the differential equation system: vectors $\dot{\delta}$ and the scalar η. Departing a state (i.e. a saddle point) happens by pushing the system in the direction of the desired subsequent state with $\dot{\delta}$ of Eq. 1. Dwell time is dependent on the magnitude of $\dot{\delta}$. The scalar η adjusts the speed at which the system evolves.

b) Construction of P: The matrix P is constructed using the rules published by Horchler et al. [4] (Eq. 5):

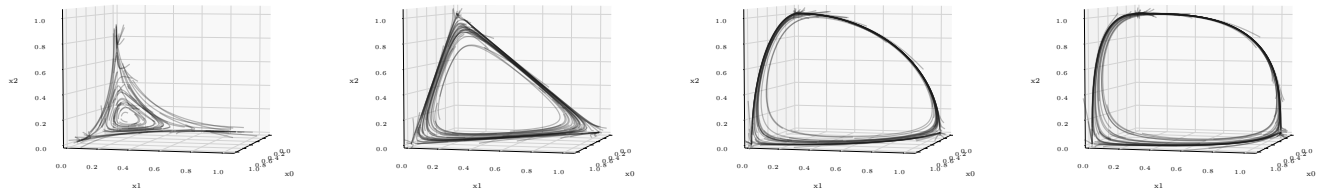
$$P_{ji} = \begin{cases} \alpha_i / \beta_j & \text{if i=j} \\ \frac{\alpha_i - \alpha_j / \nu_j}{\beta_j} & \text{if } T_{ji} = 1 \\ \alpha_i + \alpha_j / \beta_j & \text{otherwise} \end{cases} \tag{3}$$

based on the desired state connectivity matrix T, where $T_{ji} = 1$ indicates an edge from state i to j, otherwise $T_{ji} = 0$. For simplicity, we fix some parameters of the SHC-system: $\alpha_i = \alpha_0$ (uniform growth rates), $\beta_i = 1.0$ (unit state magnitudes), and $\nu_i = 1.0$ (symmetric channels). The scalar α_0 determines how quickly the system evolves, i.e. it parameterizes the time scale. For examples in this paper, $\alpha_0 = 30$.

c) Parameter γ: The γ parameter can be used to modify the shape of the heteroclinic channels. In previous work [4], $\gamma = 1$. Fig. 2 shows the effect of other values. With $\gamma = 2$ channel direction is orthogonal to the current state axis, which simplifies interpretation of input $\dot{\delta}$, which is why $\gamma = 2$ is chosen for all subsequent systems.

A. Extensions to the SHC system

The key difference between the state-machine like behavior of the SHC system and the proposed phase-state-machine is the notion of continuous transition phases. The state vector x sparsely encodes discrete states (one state per dimension), but it also implicitly encodes which transition happens and what the progress (its phase) of the transitions is. In order to specify the behavior of each state and transition independently of each other, we compute two sparse matrices that organize state and transition activations (matrix Λ), and the phases of all transition (matrix Φ). The nonlinear differential equations of the original SHC formulation [4] are further augmented with the signals $A(t)$, $B(t)$ and $E(t)$.



(a) $\gamma=0.5$ (b) $\gamma=1$ (c) $\gamma=2$ (d) $\gamma=3$

Fig. 2: Effect of γ on the attractor shape for a three-state cycle.

a) State and Transition Activation Matrix Λ: In order to sparsely encode which transition or state currently is active, we expand the state vector x into a matrix:

$$\Lambda^{\text{transitions}} = \frac{16\cdot x\otimes x\cdot|x^2|}{(x\otimes 1+1\otimes x)^4+|x|^4}\circ T \tag{4}$$

The equation is chosen such that $\Lambda_{ji}=1.0$ when the system is close to the plane spanned by states i and j, which is where a channel from saddle point i to j may be located. The outer product $x\otimes x$ forces $\Lambda_{ji}=0$ when x is close to the predecessor's or successor's state axis. Multiplication with T adds information about the *direction* of the transition by assigning the activation to either the upper or the lower triangle of the matrix. State activation is computed as the residual of transition activation, i.e. states are only active if no transition is:

$$\lambda^{\text{states}} = \frac{x^2}{\sum x^2}\cdot\left(1-\sum\Lambda^{\text{transitions}}\right) \tag{5}$$

As the diagonal of $\Lambda^{\text{transitions}}$ does not have meaningful values, we can conveniently combine all activations into a single matrix Λ:

$$\Lambda_{ji}=\begin{cases}\Lambda_{ji}^{\text{transitions}} & j\neq i\\ \lambda_i^{\text{states}} & j=i\end{cases} \tag{6}$$

State activation is scaled so that $\sum\Lambda=1.0$. This property allows us to interpret Λ as a set of coefficients for linear combination of motion generators.

b) Transition Phases Matrix Φ: The state vector sparsely encodes the discrete states (one state per dimension), but it also implicitly encodes the progress of transitions. Due to the attractor landscape, transitions happen in the plane spanned by the preceding and succeeding state vector. We therefore can compute the progress of a transition from state i to j by simply subtracting the values of the involved state axes:

$$\Phi_{ji} = \frac{|x_j|}{|x_i|+|x_j|} \tag{7}$$

which yields phase variables that grow from 0 to 1 during their associated transitions.

B. Secondary Extensions to the SHC System

Based on Λ and x we can implement behaviors that are not possible with plain SHC networks.

a) Transition velocity adjustment: Modifying transition velocities with α as proposed in [4] is quite limited: we need to ensure that $\alpha_j<\cdot\alpha_i\cdot\nu_i$ to maintain stability of the heteroclinic from state i to j. This severely limits the available range of variation. We therefore implement a different approach: the growth rate is scaled uniformly by the scalar η in Eq. 1, but only *during* transitions to maintain the stability properties close to the meta-stable saddle points. Λ allows us to specify η for each transition and state independently, using a $n\times n$ matrix A:

$$\eta=2^{\sum\Lambda\circ A} \tag{8}$$

If $A_{ji}=0$, then transition $i\to j$ will happen with "regular" speed as set by α_0. Positive values speed up exponentially, negative values slow down. The unmodified behavior can be recovered by setting $A=0$

b) Transition Biases: In order to integrate information from perception and higher-level control, we can use $\dot{\delta}$ (Eq. 1) to either push the system away from a meta-stable state, or to stabilize the state. This results in triggering or delaying the start of a transition respectively. In most practical applications though, we will want to specify the bias *towards a specific transition* instead of towards a specific successor state, as in the latter values are dependent on the current state. For example, when in state i, the bias towards itself (the ith component) should be zero to avoid shifting the saddle point along coordinate i, while at the same time we may want it to be nonzero during its predecessor state. In order to achieve independence of the biasing input from the current state, we define a matrix of stochastic biases $\dot{\Delta}\sim\mathscr{W}(B,E)$ which is parameterized by two $n\times n$ matrices that specify the mean value B_{ji} and noise value E_{ji} for each transition $i\to j$ individually. Due to the orthogonality of state vectors we can aggregate using x:

$$\dot{\delta}=\left(\dot{\Delta}\circ((1-x)\otimes x)\right)\cdot x \tag{9}$$

The mask $(1-x)$ ensures that saddle points are not shifted by $\dot{\Delta}$ accidentally. The behavior as in [4] can be recovered by setting $B=0$ and $E=\varepsilon\otimes 1$.

c) Complete system: When Eq. 1, 8 and 9 are merged, we get the following equations for computing the phase-state machine:

$$E[\dot{x}(t)]=x\circ(\alpha_0-P\cdot x^\gamma)\cdot 2^{\sum\Lambda(x)\circ A}+(B\circ((1-x)\otimes x))\cdot x \tag{10}$$

$$\sigma = (E \circ ((1-x) \otimes x)) \cdot x \tag{11}$$

C. Visualization

Visualizing an n-dimensional continuous state vector intuitively already is challenging, visualizing the evolution of two n^2-dimensional matrices over time even more so. Projection into a two-dimensional subspace does not capture the system's behavior completely, while at the same time even the smallest working examples already are four-dimensional (three state dimensions plus time). In order to achieve an intuitive visualization even with many dimensions, we adapt the concept of UML timing diagrams [12] instead, and adapt it to work with continuous activations and phases. The diagram consists of a number of lines along a time dimension. The input data for a diagram are the elements of the matrices Φ and Λ, referenced by the row and column indices j and $i \in \mathbb{Z}$. For each tuple (j,i) we can compute the function $y_{ji}(t) = i + (j-i) \cdot \Phi_{ji}(t)$, assign a corresponding line width $l_{ji}(t) = l_0 \cdot \Lambda_{ji}(t)$, and draw the resulting lines in a single diagram. In such a diagram, active states (i.e. $i = j$) are visualized as horizontal lines placed at their index value, while active transitions are visualized as continuous curves rising from preceeding state value to the succeeding state's value. The vast majority of states and transitions at any time is inactive and therefore assigned zero line width to make them invisible, avoiding visual clutter. Line width further indicates relative activation when several transitions and states are active at once. Colors are used to aid in distinguishing individual transitions. This visualization method is used throughout the rest of the paper. Important points in time are indicated by gray vertical lines.

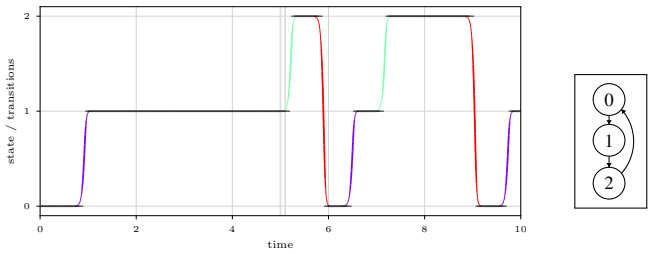
IV. Demonstrations

In this section, we will demonstrate, how various behavior patterns of state machines can be implemented with the proposed system. Some basic patterns (cycling, branching, delays) have already been demonstrated in previous work [4], [11]. We replicate those and additionally demonstrate how to implement common control patterns such as terminal states, error states, exceptions and resets. We will then demonstrate special capabilities intrinsic to the phase-state machine, such as probabilistic decisions, adjustment of transition velocity, and smooth blending of probabilistic movement primitives.

A. Staying in a State and Leaving a State

The system can be stopped from leaving a state by applying a negative $\dot{\delta}$ (Eq. 1) to all successor states. Vice versa, applying a positive $\dot{\delta}$ to a state's successor state will push the system to leave the state. In interaction strategies this is can be used e.g. to synchronize the onset of a motion.

Fig. 3 shows the state and phase evolution of a system with three states and a cyclic network ($0 \to 1$, $1 \to 2$, $2 \to 0$), and $E = 10^{-9}$. First it is locked into state 1 by setting $B_{31} = -10^{-7}$ (and all other biases zero). Then, at 5.0 s B_{31} is set to 0.1 for 0.1 s and to 0 thereafter. The applied pulse causes the system to leave the state almost immediately, which then continues to cycle through the states.



(a) (b)

Fig. 3: Example of stopping in a state, and leaving a state. By adding a negative input bias (first segment) the system stops at this state, adding a non-negative pulse at $t = 5s$ triggers the continuation of the state sequence.

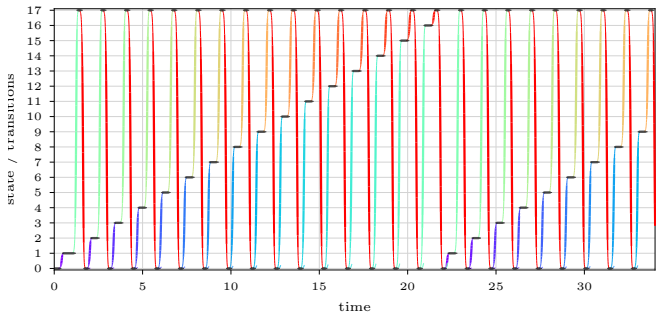


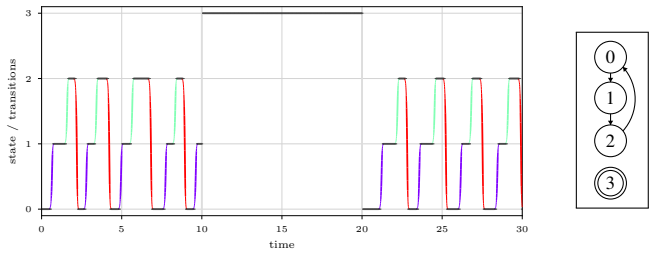
Fig. 4: Example of a system designed to branch from state 0 into states 1 to 16 and to merge back into state 17. The desired branch is selected in each iteration by setting a small $\dot{x}_{\text{bias}}$ vector.

B. Branch Out and Aggregation

Work on systems with stable heteroclinic channels already demonstrated that they can implement arbitrarily complex state machines [1]. In Fig. 4 we replicate this capability with a large number of states. State connectivity matrix T is set up so that from state 0 the state machine branches out into states 1 to 17 (fan-out of 16), which in turn all lead to state 18 (fan-in of 16) before cycling back to state 0. At the start of each cycle, B is set to prefer a different successor state. Figure 4 shows that the dynamical system has no problems implementing large fan-ins and fan-outs.

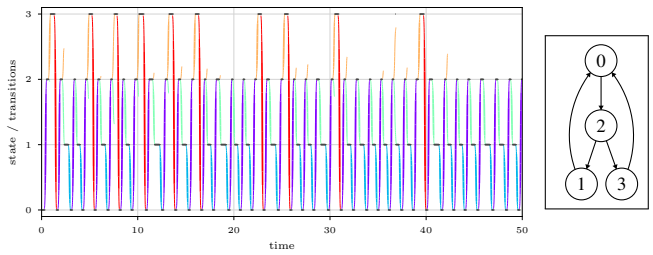
C. Excepting To Error States and Resets

Even though undesired state transitions are repelling, we can still coerce the system to transition to any state at any time by applying a large enough pulse to B. This can be used to implement resets and to except into states not reachable during normal operation (e.g. error states). Fig. 5 demonstrates this ability with a system that has an unreachable error state (state 3). By applying a pulse of $\Delta t = 20$ ms length with total area of 10.0 s to B_{30}, B_{31}, and B_{32}, we transition to this error state at $t = 10.0$ s. Likewise, we can reset the system to state 0 regardless of the state graph by applying another pulse at $t = 20.0$ s without an explicitly programmed state transition.



(a) (b)

Fig. 5: Example of excepting to error state 3 ($t=10.0$), and resetting the system to state 0 ($t=20.0$)



(a) (b)

Fig. 6: Example of implementing probabilistic decisions via bias noise. State 3 is traversed 25% of the cycles.

D. Probabilistic Decisions

An interesting aspect of the phase-state machine is its built-in capability to select transitions probabilistically. The original publications of SHC networks use stochastic noise to destabilize the saddle point equilibria over time to achieve a finite dwell time at each state [4]. But we can also use noise injected by the B input to select transitions probabilistically. Fig. 6 shows a system which branches from state 2 into states 1 and 3. The input matrix B and $E=\varepsilon\otimes 1$ are set to:

$$B=\begin{bmatrix} 0 & 10^{-9} & 0 & 10^{-9} \\ 0 & 0 & 0 & 0 \\ 10^{-4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{12}$$

$$\varepsilon=\begin{bmatrix} 0 & 3\cdot 10^{-4} & 0 & 10^{-4} \end{bmatrix} \tag{13}$$

The resulting system behavior is shown in Fig. 6. The system visits state 1 on average three times more often than state 3, due to the noise ratio $\varepsilon_1/\varepsilon_3$.

E. Slowing Down and Speeding Up Transitions

A key distinguishing feature of the phase-state machine system w.r.t. previous SHC networks and regular (discrete) state machines are the transitions of non-negligible duration. Transition periods (or phase velocities) can be adjusted for each transition individually via B. Fig. 8 shows the behavior of a system with a three-state cycle. After 3 s with $A=0$ we apply $A_{10}=-4$, $A_{21}=-7$, and $A_{02}=5$. This speeds up transition $2\to 0$ by a factor of 2^5, and slows down $0\to 1$ and $1\to 2$ by factor 2^{-4} and 2^{-7} respectively. At 20 s, 21 s and 22 s, A_{21} is set to -6, -5 and -8 respectively to demonstrate how a transition could be continuously synchronized with

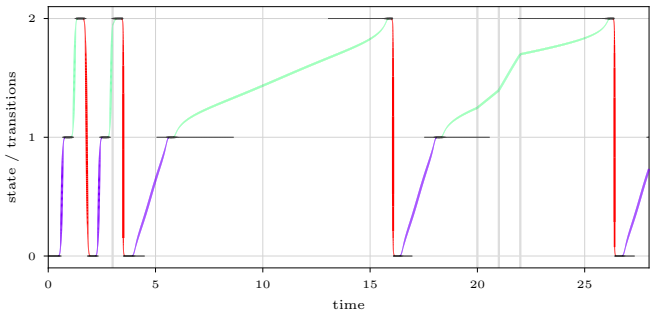


Fig. 7: Timing diagram

Fig. 8: Example of adjusting transition velocities by more than three orders of magnitude ($t=3$s), and adjusting transition velocities during a transition ($t=20$s,21 s,22s)

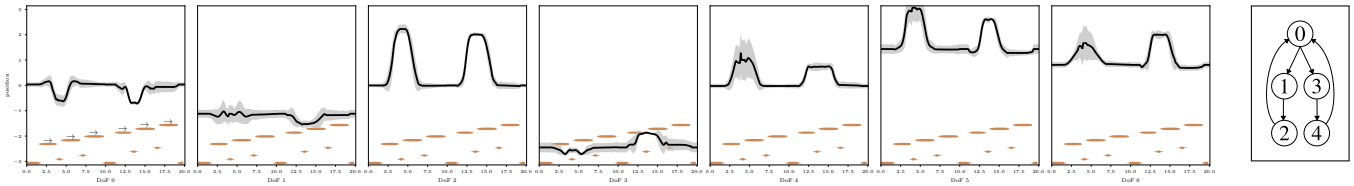
perceptual data. The system maintains stability even when transition durations span more than 3 orders of magnitude.

F. Sequencing ProMPs

Finally, we demonstrate how the phase-state machine can be applied to motion synthesis by combining it with probabilistic movement primitives (ProMP) [10], [7]. The ProMP framework is especially suited as it provides phase-parameterized trajectories, but it also provides a method to mix several concurrently active ProMPs depending on an activation value. Both phases and activations are provided by the phase-state machine. We used a Panda 7-DoF robot arm (Franka Emika) to demonstrate trajectories from an initial pose to two distinct pointing poses (10 examples each), and translated them into two probabilistic movement primitives (ProMPs [10]). For returning to the initial pose, we use the two "pointing" motions but reverse the time-phase relationship. This results in four distinct movement primitives that can be associated with specific transitions and states of a phase-state machine. We use a state graph as shown in Fig. 9b, $\alpha=20$, $E=10^{-8}$, $A=-4$, and relatively large biases to leave states quickly:

$$B=\begin{bmatrix} 0 & 0 & 10^{-3} & 0 & 10^{-3} \\ b_1 & 0 & 0 & 0 & 0 \\ 0 & 10^{-3} & 0 & 0 & 0 \\ b_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-3} & 0 \end{bmatrix} \tag{14}$$

In the experiment, $b_1=0$, $b_3=10^{-7}$ in the first five seconds and $b_1=10^{-7}$, $b_3=0$ afterwards to make the system traverse both branches. Transitions $0\to 1$ and $0\to 2$ are associated with one "pointing to" ProMP each. Transitions $1\to 2\to 0$ and $3\to 4\to 0$ are associated with ProMPs to return back to the waiting position (state 0). The return motion is split across two transitions, with the former implementing 90% of the motion and the latter the remaining 10%. This is necessary as the phase-state machine cannot implement bidirectional edges and therefore requires at least three transitions to implement a cycle. When a state is active, then its desired joint pose is determined by averaging the desired poses of all adjacent ProMPs. Fig. 9 shows the resulting behavior of the system, where the robot arm first points left



(a) Desired joint poses. (b)

Fig. 9: 7-DoF robotic arm trajectory generated the phase-state machine using ProMPs

(state 1) and then points right (state 3). Blending between subsequent ProMPs works smoothly, even when they are not matched perfectly together, e.g. when blending from state 0 to transition $0 \rightarrow 3$.

V. Discussion

The phase-state machine formalism proposed in this paper provides behaviors that could – with enough effort – also be implemented with hybrid automata. We do not *need* to switch instantly between control modes though, which makes it easy to blend motions and guarantee smooth execution. We also do not need to decide *when exactly* to switch, as the decision process too is extended over time. Further, interventions such as adaptation of execution speed can be incorporated into the system state at any time, while maintaining a consistent and complete system state, especially w.r.t. to activation and phase values. There are also some drawbacks. Transitions and states have to alternate, so "concatenating" two continuous trajectories is not possible without inserting a somewhat superfluous state in between. Also, two states cannot be mutual successors, i.e. cycling between two states is not possible. While this limitation can be circumvented by inserting a state (as done in the paper's example), it would be preferable to find a system formulation capable of implementing bidirectional edges. Another open problem is the relationship to probabilistic formalisms, specifically optimal control and bayesian inference. The system is capable of representing probabilistic policies, and it can also accumulate uncertain evidence over time. It is unclear how input signals (e.g. elements of B) can be related to conditional probabilities and hence, how Bayes-optimal decision processes can be implemented.

VI. Conclusion

The paper proposed a novel state vector decomposition method for SHC networks [4], that computes a consistent set of transition phases and activation signals. This enables a simple and effective integration with existing motion generation frameworks such as ProMPs [9]. Further, the ability to continuously modify state- and transition-specific parameters simplifies online-adaptation of desired behaviors. We demonstrated how the resulting *phase-state machine* can be used to represent discrete, markovian states as well as provide continuous phases and activation values. We demonstrated how to implement discrete behavior such as branching, error states and resets. We demonstrated the ability to adjust transition duration online and by more than three orders of magnitude, which is not possible with previous SHC networks [4]. We showed how to enact probabilistic decisions and finally, we also demonstrated how the phase-state machine can be used to govern the blending of movement primitives to create continuous, smooth motion.

References

[1] P. Ashwin and C. Postlethwaite, "On designing heteroclinic networks from graphs," *Physica D: Nonlin. Phen.*, vol. 265, pp. 26–39, 2013.

[2] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, "Learning variable impedance control," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 820–833, 2011.

[3] C. Eppner, S. Höfer, R. Jonschkowski, R. Martín-Martín, A. Sieverling, V. Wall, and O. Brock, "Lessons from the Amazon Picking Challenge: Four Aspects of Building Robotic Systems," in *Proceedings of Robotics: Science and Systems*, 2016.

[4] A. D. Horchler, K. A. Daltorio, H. J. Chiel, and R. D. Quinn, "Designing responsive pattern generators: stable heteroclinic channel cycles for modeling and control," *Bioinspiration & Biomimetics*, vol. 10, no. 2, p. 026001, 2015.

[5] R. Lioutikov, G. Neumann, G. Maeda, and J. Peters, "Learning movement primitive libraries through probabilistic segmentation," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 879–894, 2017.

[6] G. Maeda, M. Ewerton, G. Neumann, R. Lioutikov, and J. Peters, "Phase estimation for fast action recognition and trajectory generation in human–robot collaboration," *The International Journal of Robotics Research*, p. 0278364917693927, 2017.

[7] G. J. Maeda, G. Neumann, M. Ewerton, R. Lioutikov, O. Kroemer, and J. Peters, "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks," *Autonomous Robots*, vol. 41, no. 3, pp. 593–612, 2017.

[8] J. R. Medina, F. Duvallet, M. Karnam, and A. Billard, "A human-inspired controller for fluid human-robot handovers," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 324–331.

[9] A. Paraschos, G. Neumann, and J. Peters, "A probabilistic approach to robot trajectory generation," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2013, pp. 477–483.

[10] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Using probabilistic movement primitives in robotics," *Autonomous Robots*, pp. 1–23, 2017.

[11] M. I. Rabinovich, R. Huerta, P. Varona, and V. S. Afraimovich, "Transient Cognitive Dynamics, Metastability, and Decision Making," *PLoS Comput Biol*, vol. 4, no. 5, p. e1000072, 2008.

[12] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004.

[13] S. Schaal, "Dynamic Movement Primitives -A framework for Motor Control in Humans and Humanoid Robotics," in *Adaptive Motion of Animals and Machines*. Springer, Tokyo, 2006, pp. 261–280.

[14] K. W. Strabala, M. K. Lee, A. D. Dragan, J. L. Forlizzi, S. Srinivasa, M. Cakmak, and V. Micelli, "Towards Seamless Human-Robot Handovers," *Journal of Human-Robot Interaction*, vol. 2, no. 1, pp. 112–132, 2013.

[15] A. van der Schaft, "Modeling of hybrid systems," in *An introduction to hybrid dynamical systems*, ser. Lecture Notes in Control and Information Sciences. Springer, London, 2000, pp. 1–34.