2019 IEEE 58th Conference on Decision and Control (CDC)
Palais des Congrès et des Expositions Nice Acropolis
Nice, France, December 11-13, 2019

Combining Learning and Model Based Multivariable Control

G. S. Venkatesh[†] W. Steven Gray[†] Luis A. Duffaut Espinosa[‡]

***Abstract*— Artificial neural networks have traditionally been used to implement machine learning algorithms. There are, however, alternatives to these biologically inspired machine learning architectures that offer potentially lower complexity and stronger theoretical underpinnings. One such option in the context of control is based on using a generic input-output model known as a Chen-Fliess functional series. The main goal of the paper is to describe a specific architecture that can be used in the multivariable setting to combine both learning and model based control. It builds on recent work by the authors showing that a certain monoid structure underlies any recursive implementation of such a system. The method is demonstrated using a two-input, two-output Lotka-Volterra system.**

I. Introduction

Artificial neural networks have traditionally been used to implement machine learning algorithms. In the specific context of control, so called recurrent networks are a standard choice as they are capable of approximating the dynamical behavior of certain classes of systems [1], [11]–[13]. There are, however, alternatives to these biologically inspired machine learning architectures that offer potentially lower complexity and stronger theoretical underpinnings. One such option under development by the authors is based on using a generic input-output model known as a *Chen-Fliess functional series* or *Fliess operator* [3], [4], [10]. It is well known that any control-affine analytic control system in continuous-time has a Fliess operator representation. The main idea is to discretize a given operator as described in [5] and then identify its associated generating series via standard MSE estimation methods using input-output data generated by the error between the plant and a given model. Once the error system is identified, it can be used along with the model to apply, for example, predictive control [8]. The approach is distinct from model based adaptive control in that the plant is not made to track the output of the reference model, and there is no adaptation of the model. In fact, the technique can be used *without* a model as shown in [6] to provide a purely data-driven controller. But generally the availability of a model will improve performance when the error dynamics are *smaller* than the plant dynamics. One significant limitation, however, in all the previous work is that this new learning unit based on Fliess operator representations has been restricted to the single-input case. This was entirely due to the fact that constructing an efficient implementation of the learning unit for multiple inputs is nontrivial. The underlying noncommutative algebraic and combinatorial structures are not well understood.

The main goal of this paper is to describe and demonstrate one such implementation of the multivariable learning unit. It is based on work in [7] where it is shown that a certain monoid structure lies behind any recursive implementation of such a learning unit. The key result presented here is how to select a basis so that a convenient matrix representation of this monoid can be automatically generated and coded for applications. The central tool employed is a certain tree structure that can be systemically searched to construct the desired basis. The method is demonstrated using a two-input, two-output Lotka-Volterra system so that its performance can be directly compared against the existing single-input studies reported in the earlier work cited above. As expected, additional actuation dramatically improves closed-loop performance.

The paper is organized as follows. In the next section, a brief summary of the key concepts used from [5] and [7] is provided. The main results addressing the implementation of the multivariable learning unit are given in Section III. The simulation case study is presented in Section IV. The conclusions and comments regarding future work are given in the last section.

II. Preliminaries

A brief overview of some key concepts used later in the paper is given in this section. First, a class of discrete-time input-output maps used as approximators for continuous-time analytic input-output systems is defined. A more complete treatment of the topic can be found in [5]. Next, a type of learning unit based on these approximators is described. See [6], [7] for more details.

A. Discrete-Time Fliess Operator

An *alphabet* $X = \{x_0, x_1, \ldots, x_m\}$ is any nonempty and finite set of noncommuting symbols referred to as *letters*. A *word* $\eta = x_{i_1} \cdots x_{i_k}$ is a finite sequence of letters from X. The number of letters in a word η, written as $|\eta|$, is called its *length*. The empty word, $\emptyset$, is taken to have length zero. The collection of all words having length k is denoted by X^k. Define $X^* = \bigcup_{k\geq 0} X^k$. This set is a monoid under the catenation product. Any mapping $c : X^* \to \mathbb{R}^\ell$ is called a *formal power series*. Often c is written as the formal sum $c = \sum_{\eta\in X^*}(c,\eta)\eta$, where the *coefficient* (c,η) is the image of $\eta \in X^*$ under c. The set of all noncommutative formal power series over the alphabet X is denoted by $\mathbb{R}^\ell\langle\langle X\rangle\rangle$. It forms an associative $\mathbb{R}$-algebra under catenation.

Discrete-time inputs are real-valued sequences from the normed linear space

$$l_\infty^{m+1}[N_0] := \{\hat{u} = (\hat{u}(N_0), \hat{u}(N_0+1), \ldots) : \|\hat{u}\|_\infty < \infty\},$$

where $\hat{u} := [\hat{u}_0\, \hat{u}_1 \ldots \hat{u}_m]^T$, $|\hat{u}(N)| := \max_{i\in\{0,1,\ldots,m\}} |\hat{u}_i(N)|$, and $\|\hat{u}\|_\infty := \sup_{N\geq N_0} |\hat{u}(N)|$.

Definition 2.1: Given a generating series $c \in \mathbb{R}^\ell\langle\langle X\rangle\rangle$, the corresponding **discrete-time Fliess operator** is defined as

$$\hat{F}_c[\hat{u}](N) = \sum_{\eta\in X^*} (c,\eta) S_\eta[\hat{u}](N)$$

[†]Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, Virginia 23529 USA
[‡]Department of Electrical and Biomedical Engineering, University of Vermont, Burlington, Vermont 05405 USA

for any $N \geq 1$, where

$$S_{x_i\eta}[\hat{u}](N) = \sum_{k=1}^{N} \hat{u}_i(k) S_\eta[\hat{u}](k),$$

with $x_i \in X$, $\eta \in X^*$, and $\hat{u} \in l_\infty^{m+1}[1]$. By assumption, $S_\emptyset[\hat{u}](N) := 1$.

Following [9], select some fixed $u \in L_1^m[0,T]$ with $T > 0$ finite. Choose an integer $L \geq 1$, let $\Delta := T/L$ and define the sequence of real numbers

$$\hat{u}_i(N) = \int_{(N-1)\Delta}^{N\Delta} u_i(t)\, dt, \quad i = 0, 1, \ldots, m,$$

where $N \in [1, L]$. Assume $u_0 = 1$ so that $\hat{u}_0(N) = \Delta$. A truncated version of $\hat{F}_c$ will be useful,

$$\hat{y}(N) = \hat{F}_c^J[\hat{u}](N) := \sum_{j=0}^{J} \sum_{\eta \in X^j} (c, \eta) S_\eta[\hat{u}](N),$$

since numerically only finite sums can be computed. For each output of the truncated discrete-time Fliess operator above, define a column vector θ whose i-th component is given by

$$\theta_i \triangleq (c_j, \eta_i), \quad j \in \{1, \ldots, \ell\}, \tag{1}$$

where $\eta_i \in X^{\leq J} := \bigcup_{k=0}^{J} X^k$. The main assertion proved in [5] is that the class of truncated, discrete-time Fliess operators acts as a set of universal approximators with computable error bounds for their continuous-time counterparts described in [3], [4], [10]. In which case, they can be used to approximate any input-output system corresponding to a control-affine analytic state space realization. This fact is exploited in the next subsection to create a type of learning unit for data generated by such dynamical systems.

B. Learning Unit

The proposed learning unit is shown in Figure 1. It is comprised of a truncated discrete-time Fliess operator and a mean square error (MSE) estimator used to learn the parameter vectors defined in (1) from input-output data (u, y) generated by an unknown continuous-time plant. Assuming no a priori knowledge of the dynamics, the initial estimate $\hat{\theta}(0)$ is initialized to zero.[1] The output of the learning unit is given by

$$\hat{y}_p(N+1) = \hat{\theta}^T(N)\phi(N+1),$$

where $\phi(N+1)$ is the collection of iterated sums $\{S_\eta[\hat{u}](N+1)\}_{|\eta| \leq J}$ in some fixed order $\{\eta_1, \eta_2, \ldots, \eta_l\}$, where $l = \mathrm{card}(X^{\leq J}) = ((m+1)^{J+1} - 1)/m$. The estimate $\hat{\theta}(N)$ is updated by the MSE estimator using the input-output data available up to the N-th sample, that is,

$$\hat{\theta}(N) = \arg\min_{\hat{\theta}} \left\| y(N\Delta) - \hat{y}_p(N) \right\|_2.$$

The architecture is designed to learn the dynamics in real-time, thus avoiding the need to have pre-collected data for training. Using monoid representation theory, it is shown in [7] that the predicted output $\hat{y}_p(N+1)$ can be written in a purely inductive form. First define $\hat{u}_\eta(N) = \hat{u}_{i_k}(N) \cdots \hat{u}_{i_1}(N)$ for any $\eta = x_{i_k} \cdots x_{i_1} \in X^*$ and $N \geq 1$ with $\hat{u}_\emptyset(N) := 1$. In addition, let $c_u(N) := \sum_{\eta \in X^*} \hat{u}_\eta(N)\eta$. In which case,

$$\begin{aligned} \hat{y}_p(N+1) &= \hat{\theta}^T(N)\Pi(S[\hat{u}](N+1))e_1 \\ &= \hat{\theta}^T(N)\mathcal{S}(N+1)\Pi(S[\hat{u}](N))e_1, \end{aligned} \tag{2}$$

where $e_1 := [1\,0 \cdots 0]^T$, and Π is a representation map on $\mathrm{End}(\mathbb{R}^\infty)$ defined as $\Pi(S[\hat{u}](N)) = \overleftarrow{\prod_{i=1}^{N}} \mathcal{S}(i)$ with $[\mathcal{S}(i)]_{jk} = (c_u(i)\eta_k, \eta_j) = \hat{u}_\xi(i)$ and $\xi\eta_k = \eta_j$ for all $\xi, \eta_j, \eta_k \in X^*$. Of primary importance is the identification of a convenient basis for these representations so that the $\mathcal{S}(N+1)$ matrix can be inductively generated by code. This boils down to selecting a certain partial ordering for the words indexing the components of θ and ϕ. This issue is addressed in the next section.

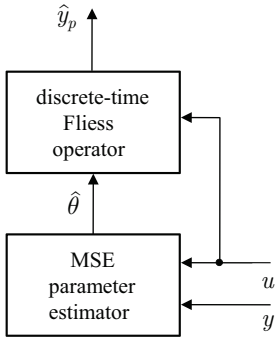


Fig. 1. Learning unit based on a discrete-time Fliess operator

III. Inductive Generation of $\mathcal{S}(N+1)$ Matrix

A partial ordering $\preceq$ is first defined on all words in X^*. For all $\zeta, \eta \in X^*$, let $\zeta \preceq \eta$ if and only if there exists a $\gamma \in X^*$ such that $\gamma^{-1}(\eta) = \zeta$, where γ^{-1} denotes the left-shift operator.

Theorem 3.1: $(X^*, \preceq)$ is a partially ordered set.

Proof: Let $\eta, \zeta, \gamma, \alpha, \beta \in X^*$. Reflexivity is trivial since $\emptyset^{-1}(\eta) = \eta$ if and only if $\eta \preceq \eta$. To prove transitivity, first observe that

$$\begin{aligned} (\eta \preceq \zeta) &\Rightarrow \exists \beta : \beta^{-1}(\zeta) = \eta \\ (\zeta \preceq \gamma) &\Rightarrow \exists \alpha : \alpha^{-1}(\gamma) = \zeta \end{aligned}$$

so that

$$((\alpha\beta)^{-1}(\gamma) = \eta) \Rightarrow \eta \preceq \gamma.$$

Therefore,

$$(\eta \preceq \zeta) \wedge (\zeta \preceq \gamma) \Rightarrow \eta \preceq \gamma.$$

To prove anti-symmetry, note that

$$(\eta \neq \zeta) \wedge (\eta \preceq \zeta) \Rightarrow \nexists \beta : \beta^{-1}(\eta) = \zeta,$$

and therefore,

$$(\eta \preceq \zeta) \wedge (\zeta \preceq \eta) \Rightarrow \eta = \zeta.$$

[1] The hat notation is used throughout to distinguish discrete-time signals from continuous-time signals, for example, $\hat{y}(N) = y(N\Delta)$. In this one instance, however, it is also used to indicate an *estimate* of parameter θ.

Hence, $(X^*, \preceq)$ is a partially ordered set. ■

The partial order $(X^*, \preceq)$ can be graphically rendered by a Hasse diagram. Starting with $\emptyset$ at the root, the Hasse diagram of $(X^*, \preceq)$ when $X = \{x_0, x_1, \ldots, x_m\}$ forms a $(m+1)$-ary infinitely branching tree. Define an injective map $R : X \longrightarrow \mathcal{C}$, where $\mathcal{C}$ is a set of colors. Color the edge between the nodes η and $x_i\eta$ with the color $R(x_i)$ in the tree. As an illustration, the tree for the case when $m = 2$ is shown in Figure 2, where $R(x_0) =$ black, $R(x_1) =$ red, and $R(x_2) =$ blue.

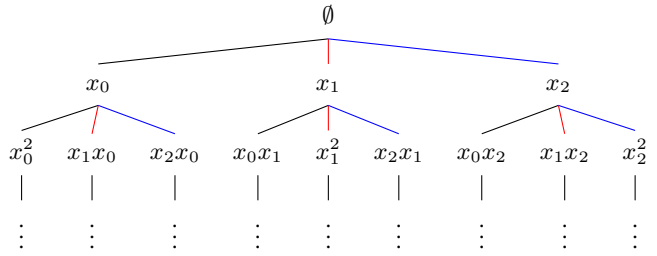


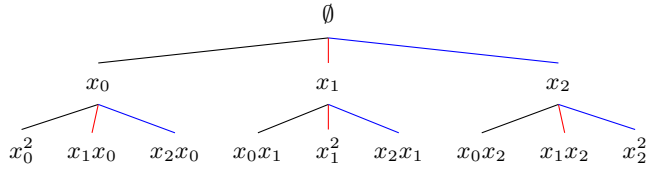
Fig. 2. Hasse diagram for $(X^*, \preceq)$ when $X = \{x_0, x_1, x_2\}$

In (2), the underlying discrete-time Fliess operator $\hat{F}_c[\hat{u}]$ has been truncated up to words of length J. Therefore, the tree is pruned at the J-th level. Next, a depth-first search (DFS) algorithm is employed to traverse the graph and generate words. The corresponding vector of words, $\chi^J(X)$, is called the *order vector* of degree J and is given by $\chi^0(X) = [\emptyset]$ and

$$\chi^{J+1}(X) = \left[\ \emptyset \ \middle|\ \chi^J(X)x_0 \ \middle|\ \chi^J(X)x_1 \ \middle|\ \cdots \ \middle|\ \chi^J(X)x_m \ \right]^T \quad (3)$$

for $J \geq 0$.

Example 3.1: The tree for words $\eta \in X^{\leq 2}$ when $X = \{x_0, x_1, x_2\}$ is given by



The DFS algorithm gives the order vector

$$\chi^2(X) = \left[\emptyset\ x_0\ x_0^2\ x_1x_0\ x_2x_0\ x_1\ x_0x_1\ x_1^2\ x_2x_1\ x_2\ x_0x_2\ x_1x_2\ x_2^2\right]^T.$$

□

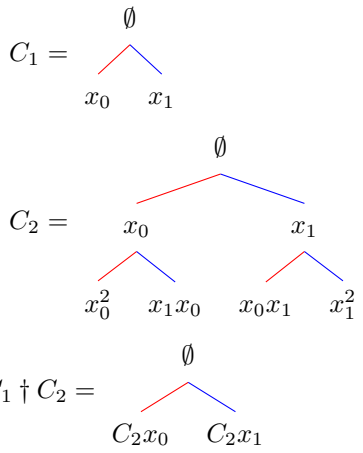
Let $\mathcal{S}^J(N+1)$ denote the matrix $\mathcal{S}(N+1)$ but truncated for words up to length J. An inductive algorithm to build such matrices is presented next. For $c \in \mathbb{R}\langle\langle X\rangle\rangle$ and υ a k-tuple of words $[\eta_1\,\eta_2\cdots\eta_k]$, let $(c,\upsilon) := [(c,\eta_1)\,(c,\eta_2)\cdots(c,\eta_k)]$. Consider the following definition.

Definition 3.1: Define C_i as the colored tree of the Hasse diagram of $(X^*, \preceq)$ up to the i-th level, that is, the $(m+1)$-ary tree with $\emptyset$ as the root and $\eta \in X^i$ as leaves of the Hasse diagram. Let $C \triangleq \{C_i : i \in \mathbb{N}_o\}$ be the set of colored trees given by $(X^*, \preceq)$ of all levels.

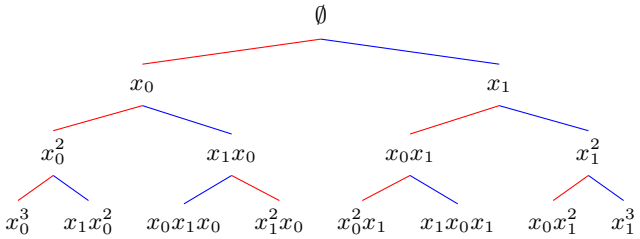
It is useful to define a product $\dagger$ on C as follows: $C_i \dagger C_j \triangleq$ tree with each leaf node $\beta \in X^i$ replaced by the tree C_j, where all the nodes of C_j are right concatenated with β. The following theorem can be proved directly by induction and basically states that there is a monoid isomorphism between $(C, \dagger)$ and the additive monoid $(\mathbb{N}_0, +)$.

Theorem 3.2: For all $C_i, C_j \in C$ it follows that $C_i \dagger C_j = C_{i+j}$.

Example 3.2: Let $X = \{x_0, x_1\}$ and define the color map R as: $R(x_0) =$ red, $R(x_1) =$ blue. Observe that



The above tree can be expanded as

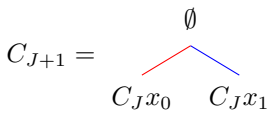


This final tree is identified as C_3 so that $C_1 \dagger C_2 = C_3$. □

Now assume that each color $R(x_i)$, $x_i \in X$, is given the weight $\hat{u}_i(N+1)$ at the discrete time instant $N+1$. Then it follows for any $\eta_j, \eta_k \in X^*$ that

$$S(N+1)_{jk} = (c_u(N+1)\eta_k, \eta_j) = \begin{cases} \text{weight of the path from } \eta_k \text{ to } \eta_j \text{ in } C_n \\ \text{where } \ n \geq |\eta_j|. \end{cases}$$

By Theorem 3.2, in the case where $X = \{x_0, x_1\}$ with color map R defined as in Example 3.2, $C_{J+1} = C_1 \dagger C_J$. That is,



Hence, from the structure of the order vector in (3) and the above tree recursion, one can deduce that the block structure of the matrix $\mathcal{S}^{J+1}(N+1)$ can be written inductively in terms of $\mathcal{S}^J(N+1)$ as:

$$\mathcal{S}^{J+1}(N+1) = \left[\begin{array}{c|c:c} \multirow{2}{*}{$(c_u, \chi^{J+1}(X))$} & 0\ \ 0\ \ 0\cdots\cdots\cdots 0 & \\ \hdashline & \mathcal{S}^J(N+1) & \mathbf{0} \\ \hdashline & \mathbf{0} & \mathcal{S}^J(N+1) \end{array}\right].$$

The above block structure can be generalized to the $(m+1)$ letter case to give the following algorithm for generating $\mathcal{S}^J(N+1)$ from $\mathcal{S}^{J-1}(N+1)$.

Pseudo-code to generate the $\mathcal{S}^J(N+1)$ matrix

$\mathcal{S}(\hat{u}(N+1), J)$
1 **if** $J = 0$ **then return** 1 // Base case
2 **else**
3 **call** $\mathcal{S}(\hat{u}(N+1), J-1)$ // Recursive call
4 $\mathcal{A} \longleftarrow \mathcal{S}(\hat{u}(N+1), J-1)$
5 $\mathcal{B} \longleftarrow$**block diag**$(\mathcal{A}, X.length)$
6 // Repeat the matrix as $m+1$ diagonal blocks
7 $\mathcal{C} \longleftarrow$**Concatenate** a row of zeros to $\mathcal{B}$
8 $\mathcal{D} \longleftarrow$**Concatenate** the column $(c_u, \chi^J(X))$ to $\mathcal{C}$
9 **return** $\mathcal{D}$

The DFS algorithm used in the computation of $\chi^J(X)$ runs as $\mathcal{O}((m+1)^J)$, which is exponential in J. The complexity of the learning unit is also exponential in J, namely, $\mathcal{O}((m+1)^{2J})$. However, if the truncation length is fixed, and the necessary order vectors are stored in a dictionary, the algorithm can be sped up, but the complexity still remains in the exponential class of order J.

Example 3.3: Let $X = \{x_0, x_1\}$ with R map as defined in Example 3.2. For $J = 2$, the words are indexed by the order vector as computed in Example 3.1. $\mathcal{S}^2(N+1)$ can be computed directly from C_2 to be

$$\mathcal{S}^2(N+1) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hat{u}_0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hat{u}_0^2 & \hat{u}_0 & 1 & 0 & 0 & 0 & 0 \\ \hat{u}_1\hat{u}_0 & \hat{u}_1 & 0 & 1 & 0 & 0 & 0 \\ \hat{u}_1 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hat{u}_0\hat{u}_1 & 0 & 0 & 0 & \hat{u}_0 & 1 & 0 \\ \hat{u}_1^2 & 0 & 0 & 0 & \hat{u}_1 & 0 & 1 \end{bmatrix}.$$

For brevity, the argument $(N+1)$ is suppressed in the elements of the matrix. The same matrix is now computed by the algorithm given above. For the base case $\mathcal{S}^0(N+1) = 1$. The matrix $\mathcal{S}^1(N+1)$ is computed using $\chi^1(X)$, which from the DFS on C_1 gives

$$\chi^1(X) = [\emptyset \quad x_0 \quad x_1].$$

From the proposed algorithm it follows that

$$\mathcal{S}^1(N+1) = \begin{bmatrix} 1 & 0 & 0 \\ \hat{u}_0 & 1 & 0 \\ \hat{u}_1 & 0 & 1 \end{bmatrix}.$$

Applying the algorithm once more to compute $\mathcal{S}^2(N+1)$:

$$\mathcal{S}^2(N+1) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hat{u}_0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hat{u}_0^2 & \hat{u}_0 & 1 & 0 & 0 & 0 & 0 \\ \hat{u}_1\hat{u}_0 & \hat{u}_1 & 0 & 1 & 0 & 0 & 0 \\ \hat{u}_1 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hat{u}_0\hat{u}_1 & 0 & 0 & 0 & \hat{u}_0 & 1 & 0 \\ \hat{u}_1^2 & 0 & 0 & 0 & \hat{u}_1 & 0 & 1 \end{bmatrix}.$$

Hence, the algorithm computes the $\mathcal{S}^2(N+1)$ matrix as expected. □

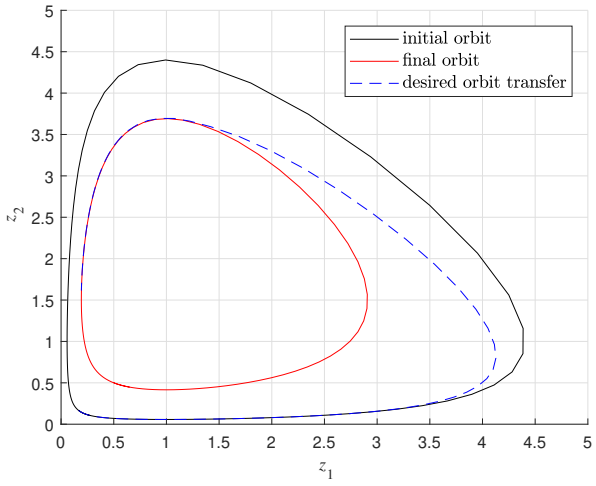


Fig. 3. Desired orbit transfer given full knowledge of the plant

IV. Case Study: Multivariable Lotka-Volterra System

The classical Lotka-Volterra model

$$\dot{z}_i = \beta_i z_i + \sum_{j=1}^{n} \alpha_{ij} z_i z_j, \quad i = 1, \ldots, n, \tag{4}$$

is often used to describe the population dynamics of n species in competition [2]. Here z_i is the biomass of the i-th species, β_i represents the growth rate of the i-th species, and the parameter α_{ij} describes the influence of the j-th species on the i-th species (normally, $\alpha_{ii} = 0$). Consider the case where a subset of system parameters $\beta_{i_j}, j = 1, \ldots, m$ in (4) can be actuated and thus viewed as inputs u_i, $i = 1, \ldots, m$. Assume some set of output functions is given

$$y_j = h_j(z), \quad j = 1, \ldots, \ell. \tag{5}$$

Since the inputs enter the dynamics linearly, it is clear that (4)-(5) constitutes a control-affine analytic state space system. In which case, the input-output system $u \mapsto y$ has an underlying Fliess operator representation F_c with generating series $c \in \mathbb{R}^{\ell}\langle\langle X \rangle\rangle$ computable from the Lotka-Volterra dynamics and a given initial condition z_0 [3], [4], [10]. Of particular interest here is the special case of a predator-prey system, which is a two dimensional Lotka-Volterra system

$$\dot{z}_1 = \beta_1 z_1 - \alpha_{12} z_1 z_2 \tag{6}$$

$$\dot{z}_2 = -\beta_2 z_2 + \alpha_{21} z_1 z_2, \tag{7}$$

where $y_1 = z_1$ and $y_2 = z_2$ are taken to be the population of prey and predator species respectively, and (4) has been re-parameterized so that $\beta_i, \alpha_{ij} > 0$. This positive system has precisely two equilibria when all the parameters are fixed, namely, a saddle point equilibrium at the origin and a center at $z_e = (\beta_2/\alpha_{21}, \beta_1/\alpha_{12})$ corresponding to periodic solutions.

Taking the system inputs in (6) and (7) to be $u_1 = \beta_1$ and $u_2 = \beta_2$, the *orbit transfer* problem as shown in Figure 3 is to determine an input to drive the system from some initial orbit to within an $\epsilon = 0.05$ neighborhood of a final orbit. The proposed controller for this task is shown in Figure 4. It is the multivariable version of the controller used for the

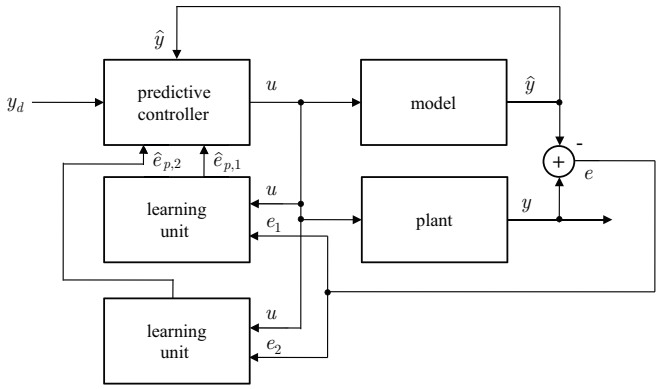


Fig. 4. Closed-loop system with MIMO predictive controller and two learning units

TABLE I

NORMALIZED RMS ERRORS FOR THE SIMULATIONS

$\Delta\alpha_{12}$	$\Delta\alpha_{21}$	δy_1	δy_2	$\|\hat{u}\|_\infty$
0	0	8.66×10^{-9}	1.25×10^{-8}	2
-5	0	0.012	0.007	2
0	-5	0.020	0.016	2
5	0	0.004	0.006	2
0	5	0.018	0.015	2
-10	0	0.016	0.012	1.5
0	-10	0.056	0.041	1.5
10	0	0.010	0.009	1.5
0	10	0.037	0.025	1.5
-20	0	0.023	0.024	0.5
0	-20	0.144	0.113	0.5
20	0	0.012	0.016	0.5
0	20	0.071	0.047	0.5
-50	0	0.092	0.096	0.5
50	0	0.010	0.028	0.5
0	50	0.062	0.095	0.5

same task in [6], [8], where only β_1 was used as an input. Its basic function is to use the learning unit in Figure 1 to learn the error system between the given model and the plant. The predicted modeling error $\hat{e}_p$ and the plant model are then employed by a one step ahead predictive controller to track the desired orbit transfer trajectory. As in practice, the model is used to generate this trajectory. The learning units were implemented using (2) and the algorithm for generating $S^J(N+1)$ with $J=3$ as described in the previous section. The expectation is a significant improvement in tracking performance over the single-input case.

The following simulations assume that all the plant's parameters are set to unity. The model's parameters are fixed and distinct from those of the plant. The sampled input was bounded by $\|\hat{u}\|_\infty$, and the positivity constraint was not enforced. For more implementation and simulation details, see [6], [8]. The tracking performance for various choices of model parameter errors is shown in Table I. For each plant parameter λ, $\Delta\lambda := (\lambda_{\text{model}} - \lambda_{\text{plant}}) * 100\%$. In addition, δy_i for $i = 1, 2$ is the RMS error normalized by the desired trajectory in each output channel. As an example, the simulation results for the cases with $+20\%$ error in α_{21} and -20% error in α_{12} are shown in Figures 5-8. The case of $\Delta\alpha_{21} = -20$, (marked red in Table I) was the extreme case as decreasing α_{21} any further resulted in the plant's response being oscillatory. The simulation results pertaining to this

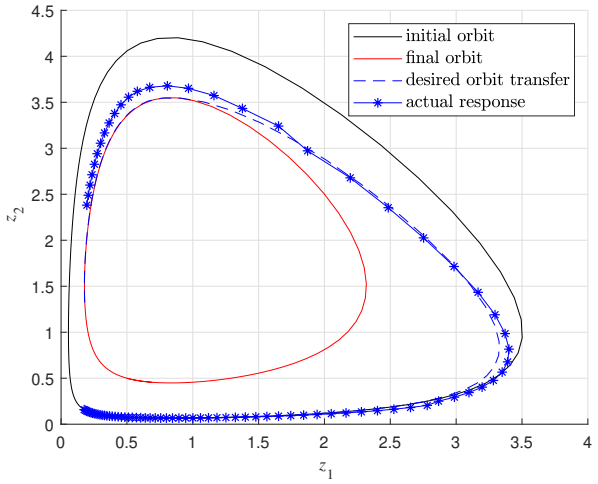


Fig. 5. Orbit transfer with +20% error in α_{21}

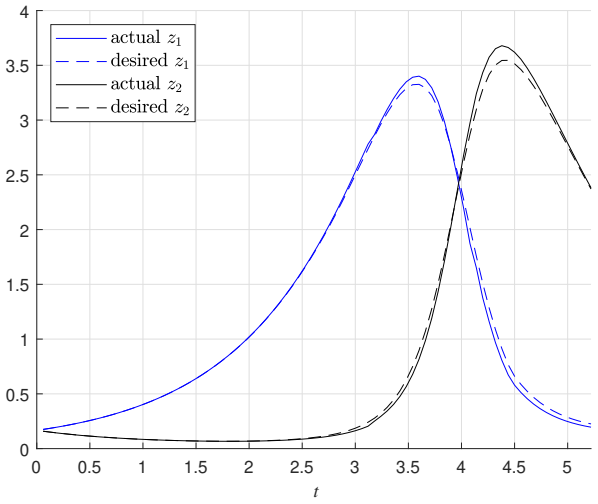


Fig. 6. State trajectories with +20% error in α_{21}

case are shown in Figures 9 and 10. Overall the simulation data demonstrated at least an order of magnitude reduction in the tracking error over the single-input cases studied in [6], [8] and significantly more robustness with respect to model parameter uncertainty. This came at the cost, however, of higher dimensional learning units, i.e., increased controller complexity.

V. CONCLUSIONS AND FUTURE WORK

It was shown in this paper how to implement a multi-variable learning unit based on a discretization of a Fliess operator for control applications. The main idea was to construct a convenient basis for a matrix representation of the underlying monoid structure using a Depth-First Search on trees which themselves form a monoid isomorphic to the additive monoid on the natural numbers. This implementation was then employed in a predictive controller to solve the orbit transfer problem for a two-input, two-output Lotka-Volterra system.

Future work will include the introduction of measurement noise in the system, exercising the method on more complex

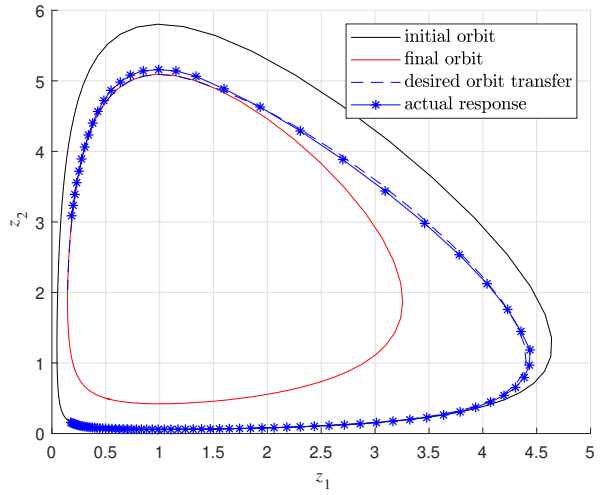


Fig. 7. Orbit transfer with -20% error in α_{12}

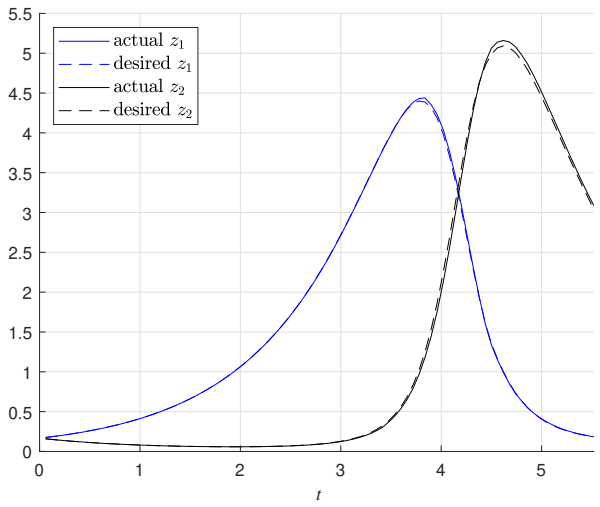


Fig. 8. State trajectories with -20% error in α_{12}

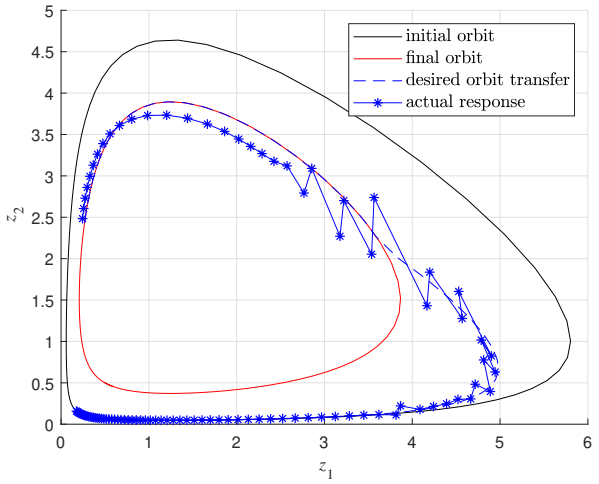


Fig. 9. Orbit transfer with -20% error in α_{21}

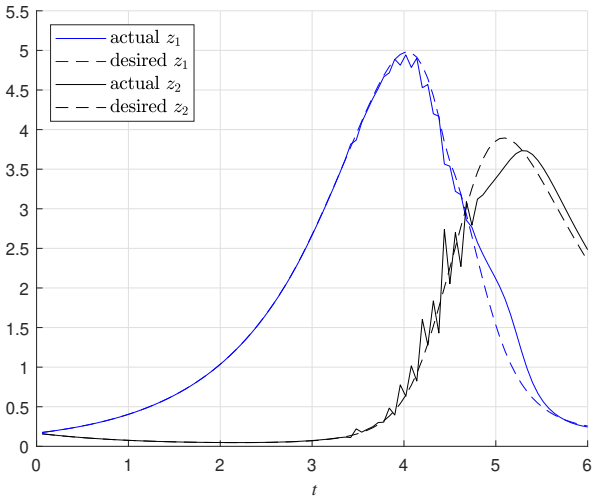


Fig. 10. State trajectories with -20% error in α_{21}

engineering plants, and identifying conditions under which closed-loop stability can be guaranteed.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under grants CMMI-1839378 and CMMI-1839387.

REFERENCES

[1] P. Baldi and K. Hornik, Universal approximation and learning of trajectories using oscillators, in *Advances in Neural Information Processing Systems*, D. Touretzky, M. Mozer, and M. Hasselmo, Eds., vol. 8, MIT Press, Cambridge, MA, 1996, pp. 451–457.

[2] E. Chauvet, J. E. Paullet, J. P. Previte, and Z. Walls, A Lotka-Volterra three-species food chain, *Mathematics Magazine*, 75 (2002) 243–255.

[3] M. Fliess, Fonctionnelles causales non linéaires et indéterminées non commutatives, *Bull. Soc. Math. France*, 109 (1981) 3–40.

[4] M. Fliess, Réalisation locale des systèmes non linéaires, algèbres de Lie filtrées transitives et séries génératrices non commutatives, *Invent. Math.*, 71 (1983) 521–537.

[5] W. S. Gray, L. A. Duffaut Espinosa, and K. Ebrahimi-Fard, Discrete-time approximations of Fliess operators, *Numer. Math.*, 137 (2017) 35–62.

[6] W. S. Gray, L. A. Duffaut Espinosa, and L. T. Kell, Data-driven SISO predictive control using adaptive discrete-time Fliess operator approximations, *Proc. 21st Inter. Conf. on System Theory, Control and Computing*, Sinaia, Romania, 2017, pp. 383–388.

[7] W. S. Gray, G. S. Venkatesh, and L. A. Duffaut Espinosa, Discrete-time Chen series for time discretization and machine learning, *Proc. 53rd 2019 Conf. on Information Sciences and Systems*, Baltimore, MD, 2019.

[8] W. S. Gray, G. S. Venkatesh, and L. A. Duffaut Espinosa, Combining learning and model based control: Case study for single-input Lotka-Volterra system, *Proc. 2019 American Control Conf.*, Philadelphia, PA, 2019, pp. 928–933.

[9] L. Grüne and P. E. Kloeden, Higher order numerical schemes for affinely controlled nonlinear systems, *Numer. Math.*, 89 (2001) 669–690.

[10] A. Isidori, *Nonlinear Control Systems*, 3rd Ed., Springer-Verlag, London, 1995.

[11] L. Jin, P. Nikiforuk, and M. Gupta, Approximation of discrete-time state-space trajectories using dynamic recurrent neural networks, *IEEE Trans. Automat. Control*, 40 (1995) 1266–1270.

[12] C. Kambhampati, F. Garces, and K. Warwick, Approximation of non-autonomous dynamic systems by continuous time recurrent neural networks, *Proc. Inter. Joint Conf. on Neural Networks*, Como, Italy, 2000, pp. 64–69.

[13] A. Schäfer and H. Zimmermann, Recurrent neural networks are universal approximators, in *Artificial Neural Networks – ICANN 2006*, S. Kollias, A. Stafylopatis, W. Duch, and E. Oja, Eds., Springer, Berlin, 2006, pp. 632–640.