DISTRIBUTED COMPUTATION IN NETWORKED SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Zak Costello

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
May 2016

DISTRIBUTED COMPUTATION IN NETWORKED SYSTEMS

Approved by:

Dr. Magnus Egerstedt
Professor, School of ECE
Georgia Institute of Technology

Dr. Anthony Yezzi
Professor, School of ECE
Georgia Institute of Technology

Dr. Patricio Vela
Associate Professor, School of ECE
Georgia Institute of Technology

Date Approved: November 2015

To my father, Mark Costello, who encouraged me to follow my dreams.

ACKNOWLEDGMENT

I would like to recognize my advisor, Dr. Magnus Egerstedt, my collaborator Dr. Justin Ruths, my reading committee, Dr Patricio Vela and Dr. Anthony Yezzi, AFOSR for funding this research, and the collective help and encouragement of the entire GRITS Lab.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The objective of this thesis is to develop a theoretical understanding of computation in networked dynamical systems and demonstrate practical applications supported by the theory. We are interested in understanding how networks of locally interacting agents can be controlled to compute arbitrary functions of the initial node states. In other words, can a dynamical networked system be made to behave like a computer? In this document, we take steps towards answering this question with a particular model class for distributed, networked systems which can be made to compute linear transformations.

In order to be specific about what is meant by computing with networked systems we begin by describing its required parts. In this work, a networked system is thought of as a dynamical system with three core components. First, there is the notion of a node. This is an entity which has a state and associated dynamics. Some examples of objects that can be modeled as nodes could be: a bird in a flock, a neuron in the brain, or a generator in a power network. Second, there must be a notion of interaction between nodes. These interactions are often described by a graph. The graph formalizes interactions between nodes using edges which indicate that the state of a node dynamically impacts its adjacent neighbors. Lastly, there needs to be some notion of external influence so that the network can be driven towards some prescribed goal. This exogenous input may act on the nodes, or the edges between nodes in the network. So these three components taken together describe a network of controllable dynamically interacting nodes.

In the field of networked control many kinds of behaviors have been synthesized by designing the interaction between neighboring nodes in a network. Some notable behaviors that designed local interaction rules enable are: distributed sensor networks [1, 2], formation control [3, 4, 5, 6, 7], and consensus [8, 9]. All of these examples in addition to implementing behaviors can be thought of as performing a computation. In formation

control, a set of aerial robots may want to reach a particular configuration. One can think of this as a mapping from initial positions to a final target formation. With distributed sensor networks, a set of hardware nodes may want to locally gather information and in a distributed manner synthesize global situational awareness. This too can be described as a mapping from sensor measurements to a global parameter of interest. Then in consensus, a set of robots rendezvous by moving towards the centroid of their neighbors. In this case, when robots move towards the centroid of their neighbors the robots compute an average of their initial states. So, another way of looking at the result of applying designed local interaction rules to a dynamical system is that they are computing a transformation. In this work, this transformation is the notion of computation.

In the following section, we will contextualize our investigation of distributed computation in networked systems and review a series of related efforts. Specifically, we examine how research in distributed computation, analog computation, and unconventional computation motivate and relate to our choice of problem formulation. In our initial efforts the focus is on computing linear transformations in a distributed manner. To this end, two fundamental questions are answered: *What global, linear transformations can be computed in finite time using edge-based interaction rules? How do we find the local rules that would compute a given linear transformation?*

This investigation is motivated by a desire to develop techniques that compute global functions of network state information. Specifically theory is developed which explains when it is possible to find weight based interaction rules which compute linear transformations of network node states. This weight based interaction paradigm is explored and the difficulty of finding these interaction rules is examined. Weighted interaction rules are found which compute particular transformations are simulated. Then these rules are then implemented on a physical network of robotic agents to illustrate their practicality. It is our intention that this technique be developed into a useful tool which can be used to synthesize global network behaviors using only local interactions.

1.1 Literature Review

One common theme when designing control and coordination mechanisms for distributed, multi-agent systems is that the information, on which decisions are based, is restricted to be shared among agents that are adjacent in the underlying information-exchange network, e.g., [10, 8, 9, 11]. As a result, local rules are needed for processing the information and coordinating the agents in the network in such a way that some global objective is achieved. Problems that fit this description can be found in a variety of applications, including power systems [12, 13, 14], formation control [3, 4, 5, 6, 7], distributed sensor networks [1, 2], smart textiles [15], and distributed optimization [16, 17]. In this thesis we take steps towards developing a general theory of local implementability/computability of such global behaviors.

As such, one key aspect of algorithm design is the definition of local interaction rules that produce desired global behaviors. An example of this are consensus algorithms for computing averages in a distributed manner. In fact, consensus plays a role in many different applications, including multi-agent robotics, distributed sensor fusion, and power network control, e.g., [9, 13, 18]. To this end, let the scalar state of each node in a network be $x_i \in \mathbb{R}$, with initial condition $x_i(t_0) = \xi_i, i = 1, \ldots, n$, where n is the number of nodes in the network. By stacking the states together in $x \in \mathbb{R}^n$,average consensus is achieved if

$$\lim_{t\to\infty} x(t) = \frac{1}{n}\begin{bmatrix} 1 & \ldots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \ldots & 1 \end{bmatrix}\xi, \tag{1}$$

where ξ is the vector containing all the initial node values. As such, the network is asymptotically computing the average, which is a global property since it relies on the state of every node.

In this thesis, a computation is defined as a mapping between states at some initial time to states at a final time by a dynamical system. So, this particular system can be thought

of as a distributed, analog, unconventional, *single purpose* computer. It is distributed because each agent can only use state information from its neighbors in the graph $\mathcal{G}$. It is analog in the sense that its outputs are real numbered values, unconventional in that the system uses a nonstandard model that goes beyond a Von Neumann architecture to perform its computation, and single purpose because it has been designed to compute one function, the average. The properties of this type of computer can be advantageous over traditional computers. Considering that a dynamic system which performs a single computation (consensus) has found so many useful applications in networked systems applications, it seems compelling to develop a more general dynamic system which is capable of performing an arbitrary computation in a distributed manner to enable even more complex and useful behaviors.

In the work that follows in this thesis we outline a more general dynamical system and develop theoretical results showing what can be computed in a distributed way. Specifically, we outline a computing model which is, distributed, analog, unconventional, and more *general purpose*. This model is used to pose the fundamental question: What can be computed in a distributed way? In this thesis, we begin to answer this question by investigating specifically what *linear* computations can be computed in a distributed way. In order to place this work in context it is necessary to explore what has been done in distributed, analog, and unconventional computing individually and discuss how those results relate to this line of inquiry.

This investigation was motivated by working to develop algorithms networked systems. While there are well developed tools which can be adapted to new network algorithms such as consensus, for new applications developing new network protocols is a research effort. In reaching an understanding of what can be computed in a distributed way we also aim to design a tool which will assist in creating new network protocols. By developing a turn key method to find weighted interaction rules which compute linear transformations the amount of effort it takes to create new behaviors for networked systems can be reduced.

One specific type of system which we emphasize in this thesis is a network of robots. Linear computations are implemented on a system of robots using weighted interaction rules. They provide a meaningful test for the practicality of implementation for such rules and show that distributed computation is both possible and useful.

1.1.1 Analog Computation

While focus of this work is theoretical in nature and primarily makes use of tools from the networked control literature, the continuous time nature of our approach makes it worthwhile to consider the analog computation literature. Computing using analog systems has a long and rich history which encompasses a diverse set of machines with different design goals. In fact, humanity's first foray into computation was done with mechanical computing machines. These include devices such as The Antikethara Mechanism built between 150 BC and 100 BC which calculated phases of the moon and other celestial information [19] and the differential analyzer in first built in 1931 which was able to numerically solve differential equations [20] and was used as a firing computer on naval ships. In modern times digital computation has been favored over analog, however, active research in this area still continues. Contributions to this field have been made across several disciplines [21].

Many different dynamic models of analog computation have been proposed including those stemming from hybrid systems, analog machine based models, Hopfield networks, and even models of space time [22, 20, 23, 24]. [22] shows that hybrid systems are capable of universal computation. Several hybrid system models are compared and the ability of these hybrid system models to simulate other computing machines is demonstrated using several methods. To our knowledge, no analog computing models are designed specifically to address computing over varying network topologies or numbers of agents. One of the contributions of our work will be to understand how to compute over arbitrary network topologies with varying numbers of agents.

While analog computers are believed not to have computational power that is greater

than their digital counterparts [25], implementation of analog computing systems in several practical applications have been shown to use significantly less power and smaller physical footprint than their digital counterparts [26]. So they may be useful in applications where footprint and power resources are limited.

1.1.2 Unconventional Computation

In order to more broadly relate this thesis to computing devices and models that are non-standard we look at the unconventional computing literature. In the field of unconventional computing, efforts are made to understand alternative ways in which computation can be achieved beyond the standard Von Neumann architecture and traditional silicon based implementation of this model. In other words, the use of any logic or dynamical system for computation which does not implement the Von Neumann architecture can be said to be unconventional. To accomplish this, both alternative logical systems and physical mediums are explored. One can think of unconventional computation research as a hedge against the physical and theoretical limits of our current computing paradigm. These unconventional computers take many forms in various mediums, such as a mechanical [27], a chemical [28], and biological[29] computing devices. These unconventional machines and models are typically pursued because they have some fundamental advantage over traditional silicon based computers.

For example in, [27], an idealized billiard ball computer is modeled. While on its face, this work seems to be a purely academic exercise, its advantage is that it implements reversible computation. Reversible computation has attracted interest because it side skirts Landauer's principle which states: "Any logically irreversible transformation of classical information is necessarily accompanied by the dissipation of at least $kTln(2)$ of heat per lost bit (about 31021J at room temperature)." [30] In other words, an AND gate (a logically irreversible transformation) necessarily requires the dissipation of at least some small fixed amount of thermal energy. So by choosing a different logical system; one entirely composed of reversible logic gates, it is possible to reduce the total amount of power required

to perform a computation.

A large body of work exists which explores realizations of these unconventional computers (e.g. [31, 29, 32]). For instance, in [32], a computing system which is biocompatable with living tissue is presented for the purpose of cancer treatment. This simple biomolecular computer can sense mRNA of disease related genes, then is able to synthesize a single strand DNA fragments in response which are known to have anti-cancer activity.

A network model for computation over arbitrary connected networks of agents is new to our knowledge of this field. It qualifies as an unconventional computation model because it uses a logical system separate than the Von Neumann architecture. Additionally, it provides the benefit that a single model can be used to determine how to perform computations over vastly different information exchange networks. The focus of this thesis is theoretical in nature, and the computing models explored will not be physically realized on unconventional hardware within the scope of our current investigation. Though, at this early stage keeping future applications in mind will allow physical realizations of the presented computing models to be undertaken when the theoretical work is sufficiently mature.

1.1.3 Distributed Computation

In this section, distributed computation is examined in the context of control of networked systems. Networked control as a discipline is devoted to questions about how to make systems of networked agents accomplish some prescribed goal with limited information. Powerful tools and and useful applications have been developed such as consensus [8, 9] and formation control [3, 4, 5, 6, 7]. Control of network systems is a broad field, but in the context of this investigation, we narrow our review of the network control literature specifically to work that performs distributed computation in some sense.

The papers presented below are most closely related to the thesis work and they also constitute the most sparse body of literature explored in this literature review. There are relatively few papers in the area of using control theory to make networked dynamical systems compute. In the general area of obtaining global information using local interactions

there are a few lines of inquiry. In [33], a fixed weighting scheme was used to compute linear transformations on networks. That work has a similar aim and takes a different discrete time approach. In a certain sense, the investigation in [34] follows this line of inquiry as well. There, quadratic invariance was used to establish whether or not a convex optimization problem exists whose solution is a decentralized implementation of a centralized feedback controller. In [35], this idea is extended to provide a practical, graph theoretic method for finding this distributed controller. Additionally, in [36] a method is presented under which consensus is computed in finite time.

The work in this thesis distinguishes itself from these other bodies of work by using a time varying weighting method, which admits the computation of global, linear transformations in finite time. Specifically, we focus on a continuous time scheme for distributed computation, over finite intervals using time-varying exogenous weight functions. We also have shown that the thesis work carves out a unique line of inquiry that fits in with both the unconventional computing and analog computing literature.

1.2 Problem Definition

In order to formally develop the problem which is explored in this thesis, we first briefly review the necessary constructs required to describe weight-based interaction rules from [37]. To this end, let V be a vertex set with cardinality n, and $E \subset V \times V$ be an edge set with cardinality m, where we insist on $(i, i) \in E,\ \forall i \in V$, as well as $(i, j) \in E \Leftrightarrow (j, i) \in E$. Let $\mathcal{G}$ be the graph $\mathcal{G} = (V, E)$, where the assumptions on E imply that $\mathcal{G}$ is undirected and contains self-loops. We moreover assume that $\mathcal{G}$ is connected. As the main purpose with $\mathcal{G}$ is to encode adjacency information in the information-exchange network, we introduce the operator sparse($\mathcal{G}$) to capture these adjacencies, and we say that an $n \times n$ matrix $M \in$ sparse($\mathcal{G}$) if $(i, j) \notin E \Rightarrow M_{ij} = 0$.

There are a number of different ways in which local interactions can be defined. In this thesis, we assume that they are given by exogenous time-varying, weights associated with

$$W(t) = \begin{bmatrix} w_{11}(t) & w_{12}(t) & 0 \\ w_{21}(t) & w_{22}(t) & w_{23}(t) \\ 0 & w_{32}(t) & w_{33}(t) \end{bmatrix}$$

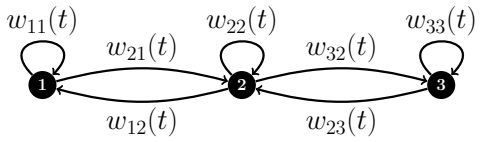


Figure 1. An illustration of what it means for $W(t) \in$ sparse($\mathcal{G}$). If $(i, j) \in E$ then $w_i j(t)$ is in the ijth entry of $W(t)$. Otherwise the ijth entry of $W(t)$ is zero. Also note that an edge $w_{ij}(t)$ is allowed to differ from $w_{ji}(t)$

the edges in the network. These weights denoted $w_{ij}(t)$ are in $L_\infty([t_0, t_f])$ where i and j indicate the originating and terminal node of the edge respectively. If $x_i \in \mathbb{R}$ is the scalar state associated with node $i \in V$, we define a local interaction as a continuous-time process

$$\dot{x}_i(t) = \sum_{j|(i,j)\in E} w_{ij}(t)x_j(t). \tag{2}$$

Note that we do not insist on $w_{ij} = w_{ji}$ even though $\mathcal{G}$ is undirected.

We focus on scalar node states, but this can be expanded trivially. If each node has more than one state, one set of interaction rules can be used for computing a function of each state. The above system illustrates that the information exchange graph governs what state information each agent has access to. Note that, as shown in Figure 1, every edge between a pair of nodes is bidirectional and each node has an associated self edge. This means that agent i has access to its own state and if agent i has access to agent j's state then agent j has access to agent i's state. However, the goal is to use these local interactions to ultimately compute a global function of the initial states. The same transformation will be computed regardless of the initial state of the system. In order to move towards this goal we can rewrite the system in ensemble form where the states associated with each agent are stacked together in a vector, $x = [x_1, \ldots, x_n]^T$.

If we stack the states together in $x = [x_1, \ldots, x_n]^T \in \mathbb{R}^n$, what we mean by *local interactions* is thus

$$\dot{x}(t) = W(t)x(t), \quad W(t) \in \text{sparse}(\mathcal{G}), \tag{3}$$

with solution

$$x(t) = \Phi(t, t_0)x(t_0), \tag{4}$$

where Φ is the state transition matrix associated with the system in (3), e.g., [38].

The purpose of the local interactions is to perform a global, linear computation. In other words, given the $n \times n$ matrix T and the initial condition $x(t_0) = \xi$, what we would like to do is find $W(t) \in \text{sparse}(\mathcal{G}),\ t \in [t_0, t_f]$, such that

$$x(t_f) = T\xi. \tag{5}$$

But, comparing this expression to (4), this simply means that what we would like is

$$\Phi(t_f, t_0) = T. \tag{6}$$

If this was indeed the case, then the local interactions, as defined through $W(t)$, would indeed compute $T\xi$ over the interval $[t_0, t_f]$ for all possible values of ξ, i.e., one can think of the network as a black box that takes ξ as the input at time t_0 and, at time t_f, returns $T\xi$ as the output.

As a final observation before we can formulate the general problem of performing global, linear computations using local interactions, we note that state transition matrix satisfies the same dynamics as (3), i.e.,

$$\frac{d\Phi(t, t_0)}{dt} = W(t)\Phi(t, t_0), \tag{7}$$

with initial condition $\Phi(t_0, t_0) = I$, where I is the $n \times n$ identity matrix. Since, the state transition matrix is what we are trying to control, its importance is emphasized in our notation by letting $\mathbf{X}(t) = \Phi(t_0, t)$. Thus the final ensemble dynamics which will be used are

$$\dot{\mathbf{X}} = W(t)\mathbf{X} \tag{8}$$

So In order to compute a given target transformation T using only local weight based interaction rules we can formalize the problem of interest succinctly as below [37].

Problem 1 (Local Computation)**.** *Given a connected graph $\mathcal{G}$ and a target linear transformation T, find $W(t) \in sparse(\mathcal{G})$ over the time horizon $t \in [t_0, t_f]$, such that*

$$\dot{\mathbf{X}} = W(t)\mathbf{X},$$

with boundary conditions $\mathbf{X}(t_0) = I$, $\mathbf{X}(t_f) = T$.

In the chapters that follow, understanding Problem 1 is the primary focus. In Chapter 2 necessary and sufficient conditions are developed for the existence of a solution to Problem 1. Chapter 3, explores the difficulty of computing solutions in detail. Chapter 4 provides several methods which are used to find solutions, simulates several example cases, and finally presents a robotic implementation of distributed computation.

CHAPTER 2

EXISTENCE CONDITIONS

In this chapter, existence conditions for computations performed using local rules over a static and undirected information-exchange network are considered. Specifically necessary and sufficient conditions for the existence of a solution to Problem 1 are developed. The local rules, once obtained, admit a decentralized implementation, where "decentralized" in this context means that each node in the network only needs to communicate state information among adjacent nodes in the network. In particular, we ask if it is possible to define local interaction laws such that $x(t_f) = T\xi$, given the linear transformation T and the initial conditions $x(t_0) = \xi$. Necessary and sufficient conditions are given for this to be possible, and they state that local interaction rules exist if and only if T has positive determinant. To this end, we start by observing that since $\mathbf{X}(t)$ is really the state transition matrix $\Phi(t, t_0)$, it is always invertible,

$$\mathbf{X}(t)^{-1} = \Phi(t, t_0)^{-1} = \Phi(t_0, t). \tag{9}$$

As a direct consequence of this, T has to be invertible for a solution to Problem 1 to exist, i.e., we need that $\det(T) \neq 0$. But, as $\mathbf{X}(0) = I$, we have that $\det(\mathbf{X}(0)) = 1 > 0$. Moreover, the determinant of a matrix depends continuously on its entries, and therefore the only way for $\det(\mathbf{X}(\tau)) < 0$ for some $\tau \in (t_0, t_f]$, there has to exist a $\tau' \in (t_0, \tau)$ such that $\det(\mathbf{X}(\tau')) = 0$. But this can not happen since $\mathbf{X}$ is always invertible. From this it directly follows that for Problem 1 to have a solution, T has to satisfy $\det(T) > 0$.

To state this fact more compactly, let $\mathrm{GL}^n_+(\mathbb{R})$ denote the set of all $n \times n$, real matrices with positive determinant. We have thus established the following necessary condition for the existence of a solution:

Lemma 1. *A solution to Problem 1 exists only if $T \in GL^n_+(\mathbb{R})$.*

Now that we have established necessary conditions for Problem 1 to have a solution, we turn our attention to sufficient conditions. And, surprisingly enough, $T \in \mathrm{GL}_+^n(\mathbb{R})$ turns out to be both necessary and sufficient for a solution to exist, which constitutes the main result in this chapter:

Theorem 1. *A solution to Problem 1 exists if and only if* $T \in GL_+^n(\mathbb{R})$.

As we have already established sufficiency, what must be shown is that whenever $\det(T) > 0$, there is a $W(t) \in \text{sparse}(G)$ that drives $\mathbf{X}$ from I to T. The remainder of this section is devoted to the establishment of this fact. However, before we can give the proof to Theorem 1, a number of supporting results are needed, involving the controllability of nonlinear, drift-free systems, i.e., systems of the form

$$\dot{x} = \sum_{i=1}^{p} g_i(x) u_i, \tag{10}$$

where $x \in \mathbb{R}^n$ is the state of the system, and $u_1, \ldots, u_p \in \mathbb{R}$ are the control inputs. For the sake of easy reference, we start by recalling Chow's Theorem, as formulated in [39], for such drift-free systems:

Theorem 2 (Chow's Theorem, e.g. [39])**.** *The system in (10) is locally controllable about a point* x_0 *if and only if*

$$dim(\overline{\Delta}(x_0)) = n, \tag{11}$$

where $\overline{\Delta}$ *is the involutive closure of the distribution span*$\{g_1, \ldots, g_p\}$.

The system is moreover controllable if it is locally controllable everywhere. And, the proof that $T \in \mathrm{GL}_+^n(\mathbb{R})$ is sufficient for Problem 1 to have a solution will hinge on showing that the dynamics, as defined through the local interaction rules in (3), is indeed controllable everywhere on $\mathrm{GL}_+^n(\mathbb{R})$. To this end, we first must rewrite the dynamics in Problem 1 on

the appropriate form. For this, we need the index matrix $\mathbb{I}_{ij} \in \mathbb{R}^{n \times n}$, which has a one at the ith row and jth column, and zeros everywhere else. The index matrix allows us to rewrite

$$\dot{\mathbf{X}} = W\mathbf{X}$$

as

$$\dot{\mathbf{X}} = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} W \odot \mathbb{I}_{ij} \right) \mathbf{X}, \tag{12}$$

where the $\odot$ symbol represents element-wise matrix product, i.e.,

$$\dot{\mathbf{X}} = \left(\begin{bmatrix} w_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} + \dots + \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & w_{nn} \end{bmatrix} \right) \mathbf{X}, \tag{13}$$

where we have surpressed the explicit dependence on t for the sake of notational ease. Rearranging the terms and letting

$$g_{ij}(\mathbf{X}) = \mathbb{I}_{ij}\mathbf{X}, \tag{14}$$

we get the drift-free matrix formulation

$$\dot{\mathbf{X}} = \sum_{i=1}^{n} \sum_{j|(i,j)\in E} g_{ij}(\mathbf{X}) w_{ij}. \tag{15}$$

To clarify, $g_{ij}(\mathbf{X})$ is a matrix whose ith row contains the jth row of $\mathbf{X}$, with the rest of the elements in the matrix equal to 0,

$$g_{ij}(\mathbf{X}) = \begin{array}{r} 1 \\ \vdots \\ i-1 \\ i \\ i+1 \\ \vdots \\ n \end{array} \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \\ \mathbf{X}_{j1} & \dots & \mathbf{X}_{jn} \\ 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}. \tag{16}$$

As a final step towards a formulation that is amenable to Chow's Theorem, let the vectorized version of g_{ij} be given by $\vec{g}_{ij} = \text{vec}(g_{ij})$, resulting in the vectorized version of (15),

$$\text{vec}(\dot{\mathbf{X}}) = \sum_{i=1}^{n} \sum_{j|(i,j)\in E} \vec{g}_{ij}(\mathbf{X}) w_{ij}. \tag{17}$$

Once a drift free form of the system dynamics are derived, controllability can be evaluated using Chow's Theorem. The first order of business towards establishing controllability of this system is the derivation of the Lie brackets for the system in (17).

Lemma 2.

$$[\vec{g}_{ij}(\mathbf{X}), \vec{g}_{kl}(\mathbf{X})] = \begin{cases} -\vec{g}_{il}(\mathbf{X}) & \textit{if } j = k,\ i \neq l \\ \vec{g}_{kj}(\mathbf{X}) & \textit{if } i = l,\ j \neq k \\ \mathbf{0} & \textit{otherwise} \end{cases} \tag{18}$$

Proof. The Lie bracket $[\vec{g}_{ij}(\mathbf{X}), \vec{g}_{kl}(\mathbf{X})]$ is given by

$$[\vec{g}_{ij}(\mathbf{X}), \vec{g}_{kl}(\mathbf{X})] = \frac{\partial \vec{g}_{kl}(\mathbf{X})}{\partial \text{vec}(\mathbf{X})} \vec{g}_{ij}(\mathbf{X}) - \frac{\partial \vec{g}_{ij}(\mathbf{X})}{\partial \text{vec}(\mathbf{X})} \vec{g}_{kl}(\mathbf{X}), \tag{19}$$

Substitution of (14) into (19), the above expression yields

$$\frac{\partial(\text{vec}(\mathbb{I}_{kl}\mathbf{X}))}{\partial \text{vec}(\mathbf{X})} \text{vec}(\mathbb{I}_{ij}\mathbf{X}) - \frac{\partial(\text{vec}(\mathbb{I}_{ij}\mathbf{X}))}{\partial \text{vec}(\mathbf{X})} \text{vec}(\mathbb{I}_{kl}\mathbf{X}), \tag{20}$$

which can be rewritten, using the Kronecker product, as

$$\frac{\partial((I \otimes \mathbb{I}_{kl})\text{vec}(\mathbf{X}))}{\partial \text{vec}(\mathbf{X})} (I \otimes \mathbb{I}_{ij})\text{vec}(\mathbf{X}) - \frac{\partial((I \otimes \mathbb{I}_{ij})\text{vec}(\mathbf{X}))}{\partial \text{vec}(\mathbf{X})} (I \otimes \mathbb{I}_{kl})\text{vec}(\mathbf{X}) \tag{21}$$

Taking the above derivatives yields

$$(I \otimes \mathbb{I}_{kl})(I \otimes \mathbb{I}_{ij})\text{vec}(\mathbf{X}) - (I \otimes \mathbb{I}_{ij})(I \otimes \mathbb{I}_{kl})\text{vec}(\mathbf{X}). \tag{22}$$

Using the mixed product property of the Kronecker product, (19) can be further simplified to

$$(I \otimes \mathbb{I}_{kl}\mathbb{I}_{ij})\text{vec}(\mathbf{X}) - (I \otimes \mathbb{I}_{ij}\mathbb{I}_{kl})\text{vec}(\mathbf{X}), \tag{23}$$

i.e., the Lie bracket in (19) becomes

$$[\vec{g}_{ij}(\mathbf{X}), \vec{g}_{kl}(\mathbf{X})] = \text{vec}(\mathbb{I}_{kl}\mathbb{I}_{ij}\mathbf{X}) - \text{vec}(\mathbb{I}_{ij}\mathbb{I}_{kl}\mathbf{X}). \tag{24}$$

Now, using the fact that, $\mathbb{I}_{ij}\mathbb{I}_{kl} = \mathbb{I}_{il}$ if $j = k$ and $\mathbb{I}_{ij}\mathbb{I}_{kl} = 0$ otherwise, we can break down (24) into 3 cases:

First if $j = k$ and $i \neq l$ we get

$$[\vec{g}_{ij}(\mathbf{X}), \vec{g}_{kl}(\mathbf{X})] = -\text{vec}(\mathbb{I}_{il}\mathbf{X}) = -\vec{g}_{il}(\mathbf{X}). \tag{25}$$

The second case occurs when $i = l$ and $j \neq k$, in which case

$$[\vec{g}_{ij}(\mathbf{X}), \vec{g}_{kl}(\mathbf{X})] = \text{vec}(\mathbb{I}_{kj}\mathbf{X}) = \vec{g}_{kj}(\mathbf{X}). \tag{26}$$

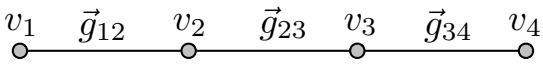
Otherwise, the Lie bracket is $\mathbf{0}$, and the lemma follows. □

Now that Lie brackets can be computed in general for this problem, we must determine if the involutive closure of the distribution associated with the system in (17) contains enough independent vector fields for local controllability. To help with this determination, we provide the following lemma.

Lemma 3. *If node i is path-connected to node j, then $\vec{g}_{ij}(\mathbf{X})$ is in the distribution $\overline{\Delta}(\mathbf{X})$.*

Proof. That node i is path-connected to node j means that there is a path through adjacent nodes in the graph G that starts at node i and ends at node j. Assume that the path goes through the nodes $N_1, \ldots, N_q$, i.e., N_1 is adjacent to N_2, N_2 is adjacent to N_3, and so forth, while $N_1 = i$ and $N_q = j$. Since these nodes are adjacent, we, by definition, have that $\vec{g}_{N_1N_2}, \vec{g}_{N_2N_3}, \ldots, \vec{g}_{N_{q-1}N_q} \in \Delta(\mathbf{X})$.

The involutive closure contains every possible Lie bracket that can be recursively created from elements $\Delta(\mathbf{X})$, which implies that the problem is to create $\vec{g}_{ij}$ from some combination of Lie brackets from elements in $\Delta(\mathbf{X})$. And, from Lemma 2, we know that $[\vec{g}_{N_1N_2}, \vec{g}_{N_2N_3}]$ is equal to $-\vec{g}_{N_1N_3}$. Applying Lemma 2 again gives $[-\vec{g}_{N_1N_3}, \vec{g}_{N_3N_4}] = \vec{g}_{N_1N_4}$.



$$[\vec{g}_{12}, \vec{g}_{23}] = -\vec{g}_{13}$$
$$[-\vec{g}_{13}, \vec{g}_{34}] = \vec{g}_{14}$$

Figure 2. An example of the construction in the proof of Lemma 3 with node i and j being represented by v_1 and v_4, respectively.

This procedure can be repeated until we arrive at one of two possible cases. If q is even, the result is $[-\vec{g}_{N_1N_{q-1}}, \vec{g}_{N_{q-1}N_q}] = \vec{g}_{N_1N_q}$. If q is odd we get $[\vec{g}_{N_1N_{q-1}}, \vec{g}_{N_{q-1}N_q}] = -\vec{g}_{N_1N_q}$. In either case, we are able to construct $\vec{g}_{N_1N_q}$ from previous Lie brackets, as shown in Figure 2. And, as $N_1 = i$ and $N_q = j$, we have $\vec{g}_{ij} \in \overline{\Delta}(\mathbf{X})$. □

Additionally the linear independence of vector fields is needed in order to establish controllability. To this end the following lemma is presented.

Lemma 4. *$\{\vec{g}_{uv}(\mathbf{X}) \; \forall \; u, v \in V\}$ is a set of linearly independent vectors.*

Proof. By definition, $\vec{g}_{uv}(\mathbf{X}) = vec(\mathbb{I}_{uv}\mathbf{X})$. This definition can be expanded to $vec(\mathbb{I}_{uv}\mathbf{X}) = (\mathbf{X}^T \otimes I_n)vec(\mathbb{I}_{uv})$. Concatenating all possible vectors resulting from combinations of vertices,

$$(\mathbf{X}^T \otimes I_n)\begin{bmatrix} vec(\mathbb{I}_{11}) & vec(\mathbb{I}_{21}) & \dots & vec(\mathbb{I}_{nn}) \end{bmatrix} \tag{27}$$

which can be further simplified to

$$(\mathbf{X}^T \otimes I_n)I_{n^2} \tag{28}$$

Taking the determinant of this expression yields

$$det(\mathbf{X}^T \otimes I_n) = det(\mathbf{X}^T)^n \tag{29}$$

Because $\mathbf{X} \in GL_n^+(\mathbb{R})$ the determinant of X is always positive and therefore we can write

$$det(\mathbf{X}^T)^n \neq 0 \tag{30}$$

This implies that the set of vectors $\{\vec{g}_{uv}(\mathbf{X}) \forall u, v \in V\}$ is linearly independent. □

To establish that the system is controllable on $\mathrm{GL}_+^n(\mathbb{R})$, $\overline{\Delta}(\mathbf{X})$ must have rank n^2 everywhere on this set, which is the topic of the next lemma.

Lemma 5. *If $\mathcal{G}$ is connected then $\overline{\Delta}(\mathbf{X})$ has dimension n^2 if and only if rank$(\mathbf{X}) = n$.*

Proof. To prove this lemma, we need to show that the implication goes both ways. Assume first that $\dim(\overline{\Delta}(\mathbf{X})) = n^2$. If $\mathcal{G}$ is connected then, by Lemma 3 and Lemma 4 the set $\{g_{ij}(\mathbf{X}) | i, j \in V\}$ is in $\bar{\Delta}(\mathbf{X})$ and is linearly independent. Therefore,

$$\overline{\Delta}(\mathbf{X}) = \mathrm{span}\{\vec{g}_{ij},\ \forall (i,j) \in V \times V\}. \tag{31}$$

For the purpose of the proof, it is convenient to go back to the matrix formulation, and we recall that $\vec{g}_{ij} = vec(g_{ij})$. As such, we will use the matrix form g_{ij} to construct $\mathbf{X}$. And, since the goal is to form a matrix with rank n, only n linearly independent matrices are needed. So, we arbitrarily choose to form $\mathbf{X}$ from the "diagonal" set $\{g_{11}, g_{22}, \ldots, g_{nn}\}$. Using the fact that $g_{ij} = \mathbb{I}_{ij}\mathbf{X}$, we can write,

$$\sum_{i=1}^{n} g_{ii} = \sum_{i=1}^{n} \mathbb{I}_{ii}\mathbf{X},$$

which simplifies to

$$\sum_{i=1}^{n} g_{ii} = \mathbf{X}. \tag{32}$$

g_{ii} is a matrix with one nonzero row at row i. The nonzero rows of each g_{ii} are linearly independent. And, since $\mathbf{X}$ is composed of n linearly independent rows, $\mathrm{rank}(\mathbf{X}) = n$, and the first implication follows. Next, we must show that

$$\mathrm{rank}(\mathbf{X}) = n \Rightarrow \dim(\overline{\Delta}(\mathbf{X})) = n^2, \tag{33}$$

which we do by contradiction. Using the expression $g_{ij} = \mathbb{I}_{ij}\mathbf{X}$, n^2 matrices can be formed from $\mathbf{X}$. Let us assume that they are not linearly independent. This implies that there exists a set of coefficients α_{ij} such that, for some (k, l),

$$\sum_{(i,j)\neq(k,l)} g_{ij}\alpha_{ij} = g_{kl}. \tag{34}$$

Since $\mathbf{X}$ has full rank, $\mathbf{X}$ can be removed from (34) based on the fact that $g_{ij} = \mathbb{I}_{ij}\mathbf{X}$, yielding

$$\sum_{(i,j)\neq(k,l)} \mathbb{I}_{ij}\alpha_{ij} = \mathbb{I}_{kl}. \tag{35}$$

By definition of the index matrix, (35) cannot be true, since every matrix in the sum on the left has a value of zero where $\mathbb{I}_{kl}$ has value of 1. Therefore, we have reached a contradiction and can conclude that $\dim(\overline{\Delta}(\mathbf{X})) = n^2$. □

Since $\mathbf{X}$ is really a state transition matrix, i.e., it is indeed invertible (with $\text{rank}(\mathbf{X}) = n$), the system in (15) is locally controllable everywhere on $\text{GL}^n_+(\mathbb{R})$ as long as the underlying graph G is connected:

Theorem 3. *The system*

$$\dot{\mathbf{X}} = W\mathbf{X},\ W \in \mathit{sparse}(\mathcal{G})$$

is locally controllable everywhere on $GL^n_+(\mathbb{R})$ if $\mathcal{G}$ is connected.

Theorem 3 and Lemma 1 give us all the ammunition needed to prove the main result in this paper, namely Theorem 1:

Proof of Theorem 1. Lemma 1 tells us that a solution only exists if $T \in \text{GL}^n_+(\mathbb{R})$, so what remains is to establish that this is indeed sufficient. Hence, assume that $T \in \text{GL}^n_+(\mathbb{R})$. Since $I \in \text{GL}^n_+(\mathbb{R})$, and $\text{GL}^n_+(\mathbb{R})$ is connected [40], there is a continuous curve of matrices in $\text{GL}^n_+(\mathbb{R})$ that connects I and T. And, by Theorem 3, every point along the path connecting I and T is locally controllable. The system being drift-free moreover implies that it can flow along this curve, e.g., [41]. Therefore, a solution to Problem 1 exists if $T \in \text{GL}^n_+(\mathbb{R})$. □

2.1 Consequences and Extensions of Theorem 1

Theorem 1 provides tight conditions for which global linear transformations can be computed using local weight based interaction rules. This result has several implications for the practical use of weight based interaction rules for real problems. In this section we discuss some of the consequences of Theorem 1. Additionally, ways to work around the restrictions it places on the set of possible computations are provided.

2.1.1 T^{-1} can be directly Computed

Suppose that weighted interaction rules $W(t)$ are found which compute a target transformation T on the initial states of the nodes in an information exchange graph $\mathcal{G}$. Those weighted interaction rules can be trivially modified so that the same network can compute T^{-1}. Noting that $\mathbf{X}(t) = \Phi(t, t_0)$ is the state transition matrix, we can rewrite our original system as

$$\begin{aligned} \dot{\Phi}(t, t_0) &= W(t)\Phi(t, t_0), \\ \Phi(t_0, t_0) &= I, \\ \Phi(t_f, t_0) &= T. \end{aligned} \tag{36}$$

The state transition matrix can be inverted as, $\Phi(t, t_0)^{-1} = \Phi(t_0, t)$. So, $\Phi(t_f, t_0) = T$ implies $\Phi(t_0, t_f) = T^{-1}$ then $\mathbf{X}^{-1}(t_f) = T^{-1}$. Computing T^{-1} now only requires that the original differential equation be run backwards starting at t_f and run back to t_0. This can be accomplished as below,

$$\begin{aligned} \dot{\Phi}(t, t_f) &= -W(t_f - t)\Phi(t, t_f), \\ \Phi(t_f, t_f) &= I, \\ \Phi(t_0, t_f) &= T^{-1}. \end{aligned} \tag{37}$$

This means given a set of weighted interaction rules $W(t)$ that compute the target transformation T, the weighted interaction rules that compute T^{-1} are $-W(t_f - t)$.

2.1.2 Finite Time Consensus Is Impossible

As a consequence of Lemma 1 it is impossible to use local rules, as understood in this paper, to achieve consensus in finite time. This follows directly from the fact that the consensus computation is given by the linear map

$$T_{cons} = \frac{1}{n}\mathbf{1}^T\mathbf{1}, \tag{38}$$

where $\mathbf{1}$ is a vector of length n, with all entries equal to one. And,

$$\text{rank}(T_{cons}) = 1,$$

i.e., $\det(T_{cons}) = 0$. Note, of course, that asymptotic consensus is possible, e.g. [8, 9, 11, 5]. To formalize this point the following corollary is provided:

Corollary 1. *There is no solution to Problem 1 which admits finite time consensus.*[1,2]

Proof. Consensus dynamics of the form $\dot{x} = -Lx$ asymptotically approach a solution, $x(t) = \frac{1}{n}\mathbf{1}^T\mathbf{1}x_0$ where $\mathbf{1}$ is a vector in $\mathbb{R}^n$ composed of all ones. In order for our scheme to compute the consensus of states in finite time, we must choose the target transformation to be $T = \frac{1}{n}\mathbf{1}^T\mathbf{1}$. Since $det(\frac{1}{n}\mathbf{1}^T\mathbf{1}) = 0$, by Theorem 1 there is no solution to Problem 1 that will allow us to reach consensus in finite time. □

2.1.3 A Single Node Can Compute Any Linear Function of Network States

If we return to the consensus problem, we have already established that T_{cons} in (38) is not computable in finite time using local rules. However, consider instead the transformation

$$T_{cons2} = \begin{bmatrix} 1/n & 1/n & \cdots & 1/n \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{39}$$

We have

$$\det(T_{cons2}) = \frac{1}{n} \tag{40}$$

and, as such, it is computable using local rules. In this case, the network average is only computed by a single node (node 1 in this case), while the remaining nodes return to their initial values at the end of the computation. This can in fact be generalized to any scalar,

[1] Note that this applies to any agreement across the nodes, i.e., not only to average consensus.

[2] It is possible to reach consensus asymptotically by choosing an $T = e^{Lt}$ where L is the graph laplacian and t is an arbitrary time. This choice of T would be computed in succession for all time.

non-zero, linear map $\ell : \mathbb{R}^n \to \mathbb{R}$ through

$$T_\ell \xi = \begin{bmatrix} \ell(\xi) \\ \xi_2 \\ \vdots \\ \xi_n \end{bmatrix},$$

where we have assumed that $\ell(\xi)$ depends on ξ_1. If not, simply pick another node in the network that ξ does depend on, as the node where the computation takes place. The point with this is that *it is possible to compute any scalar, non-zero, linear map as long as the computation only has to take place at a single node.*

2.1.4 Practical Computation of Any Linear Transform

In previous sections we discussed how Theorem 1 limits what transformations can be computed in finite time. In this section, we show that with a doubling of the state space and a modification of problem structure, Theorem 1 can be satisfied and any desired target linear transformation, T_{any}, can be realized. Let $T_{any} \in \mathbb{R}$ be a matrix with any determinant. Now, consider a system with n agents where each agent i has two states instead of one. One of these states keeps track of the computational output $\gamma_i(t)$, and the other is memory state $\mu_i(t)$. The original state vector can be rewritten as

$$x(t) = \begin{bmatrix} \gamma(t) \\ \mu(t) \end{bmatrix}, \tag{41}$$

where $\gamma(t)$ and $\mu(t)$ are vectors containing each agent's output state and memory state respectively. Let the initial states of this matrix be given by

$$x(t_0) = \begin{bmatrix} \xi \\ \mathbf{0} \end{bmatrix} \tag{42}$$

where ξ is the exogenous input into the network computation. Since each node is associated with only one state, a connected augmented information exchange graph is required. This graph is defined by taking the disjoint union of $\mathcal{G}$ with itself, then adding edges between

nodes controlled by an individual agent. This new graph is termed $\mathcal{G}_{aug}$. An example of transforming a graph $\mathcal{G}$ into its augmented graph $\mathcal{G}_{aug}$ is shown in figure 3.

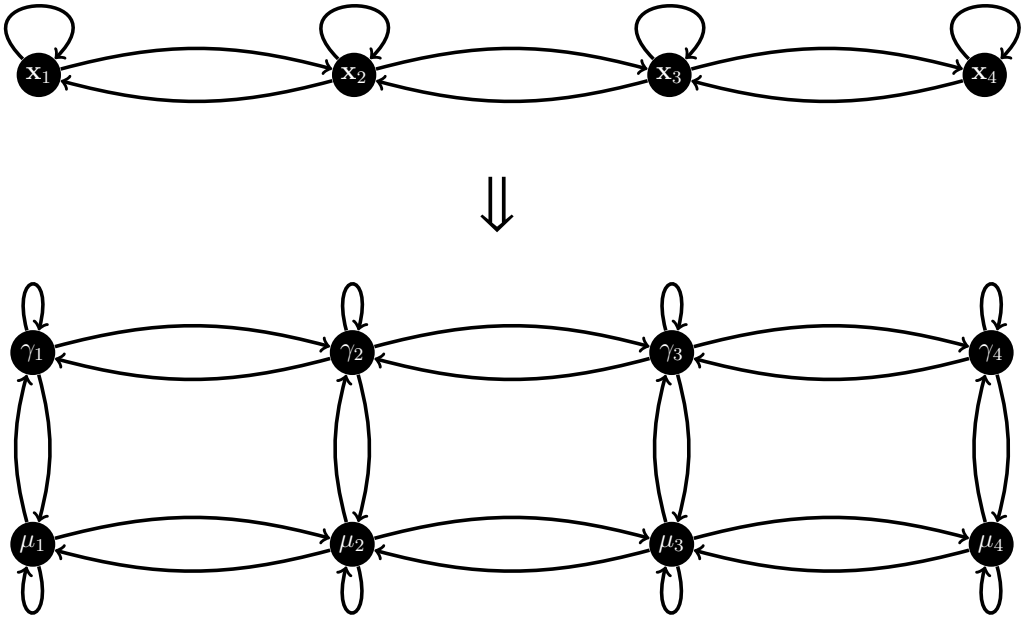


Figure 3. An example showing how the standard interaction exchange graph $\mathcal{G}$ is transformed into the augmented information exchange graph, $\mathcal{G}_{aug}$. The augmented graph is formed by taking the graph Cartesian product, $\mathcal{G} \square \mathcal{G}_{line} = \mathcal{G}_{aug}$, where $\mathcal{G}_{line}$ is the two node line graph. Both graphs have nodes that are labeled with their associated state. $\mathcal{G}_{aug}$ has two nodes associated with each agent. Agent i controls the state γ_i and μ_i which are the output state and memory state respectively.

This new graph results in a new sparse weight matrix $W(t)$ with dimension $2n \times 2n$. This matrix will take the form

$$W(t) = \begin{bmatrix} W_{11}(t) & W_{12}(t) \\ W_{21}(t) & W_{22}(t) \end{bmatrix} \tag{43}$$

where $W_{11}(t), W_{22}(t) \in sparse(\mathcal{G})$ and $W_{12}(t), W_{21}(t)$ are diagonal weight matrices. In order to phrase a problem in the form of Problem 1 such that a solution can be found, T must be chosen to be in $GL_+^{2n}(\mathbb{R})$. By allowing the target transformation to take the form below,

$$T = \begin{bmatrix} T_{any} & A \\ B & C \end{bmatrix} \tag{44}$$

where $A, B, C \in \mathbb{R}^n$ are chosen so the block matrix T has positive determinant this requirement is satisfied. Putting together all of the pieces we can define a problem with an

augmented state space which has a solution for any target transformation T_{any}.

Problem 2 (Augmented Local Computation)**.** *Given a connected graph $\mathcal{G}$ and a target linear transformation T_{any}, find $W(t) \in sparse(\mathcal{G}_{aug})$ over the time horizon $t \in [t_0, t_f]$, such that*

$$\dot{\mathbf{X}} = \begin{bmatrix} W_{11}(t) & W_{12}(t) \\ W_{21}(t) & W_{22}(t) \end{bmatrix} \mathbf{X},$$

with boundary conditions

$$\mathbf{X}(t_0) = I, \quad \mathbf{X}(t_f) = \begin{bmatrix} T_{any} & A \\ B & C \end{bmatrix}.$$

$\mathbf{X}(t)$ in the augmented problem has dimension $2n \times 2n$. The resulting computation performed when Problem 2 is solved is

$$\begin{bmatrix} \gamma(t) \\ \mu(t) \end{bmatrix} = \begin{bmatrix} T_{any} & A \\ B & C \end{bmatrix} \begin{bmatrix} \xi \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} T_{any}\xi \\ B\xi \end{bmatrix} \tag{45}$$

Using all of the augmented variables a solvable problem in the form of 1 was formulated where arbitrary linear transformations can be computed through the output node at the final time $\gamma(t_f) = T_{any}\xi_0$. At t_f, the value associated with the memory state $\mu(t)$ is discarded. Therefore any linear transformation can be computed at the cost of augmenting the state space by solving Problem 2. In this section, we showed that by doubling the state space, the limit of a transformation T having positive determinant in Theorem 1 can be overcome. We have illustrated that it is possible to compute a transformation $T_{any} \in \mathbb{R}^{n \times n}$ which can have any determinant by solving Problem 2.

2.1.5 Introducing Nonlinearities

Fundamentally, using weights to perform computations on networks with dynamics defined by $\dot{\mathbf{X}} = W\mathbf{X}$ can only compute linear transforms. However, the inputs to the system can

be chosen to be nonlinear. If it is assumed that agents can compute nonlinear functions of their own states and then execute linear computations, then nonlinear computations can be synthesized. To illustrate this fact, Let the initial node states ξ be given by

$$\xi = \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_n \end{bmatrix} = \begin{bmatrix} f_1(x_1(t_0)) \\ \vdots \\ f_n(x_n(t_0)) \end{bmatrix} = f(x(t_0)) \tag{46}$$

where $f_i(x_i(t_0))$ are arbitrary nonlinear functions of the state of node i at time t_0. These initial exogenous inputs when coupled with weights $W(t)$ with $t \in [t_0, t_f]$ allow the dynamical system $\dot{\mathbf{X}} = W\mathbf{X}$ run until time t_f and compute

$$x(t_f) = T f(x(t_0)). \tag{47}$$

This is a linear combination of nonlinear functions. We can go further by allowing multiple computations to be cascaded together. Consider now a set of weighting functions that compute several different transformations, $\{W_1(t), \ldots, W_p(t)\}$ which when used on the dynamical system $\dot{\mathbf{X}} = W_i(t)\mathbf{X}$ compute $T_1, \ldots, T_p$ respectively. Now consider also that after each computation that occurs in sequence, each node applies a nonlinear function to the output and feeds the result into the next computation. Let these nonlinear computations be noted $f_1, \ldots, f_p$. The resulting computation can be written as

$$x(t_f) = T_p f_p(T_{p-1} f_{p-1}(\ldots(T_1 f_1(x(t_0))))) \tag{48}$$

Any nonlinear computation which can be decomposed in this way can be computed by this framework.

CHAPTER 3

ON THE DIFFICULTY OF FINDING SOLUTIONS

In Chapter 2, conditions for the existence of solutions to Problem 1 were established. Just because a solution exists, doest not mean methods for finding solutions can be easily developed. Given that a target transformation is chosen which satisfies the existence condition, $T \in GL_n^+(\mathbb{R})$, the difficulty of finding solutions is explored. First, the space of positive determinant matrices is briefly illustrated using plots for 2×2 and 3×3 matrices. This is intended to provide an appreciation of the non convexity of the space of positive determinant matrices to the reader. Additionally, a differential geometric perspective on control synthesis is provided which explains why more sparse interaction exchange graphs may be, in some sense, harder to compute with. Finally we conclude with the implications of Brockett's theorem on control synthesis.

3.1 What Does $GL_+^n(\mathbb{R})$ Look Like?

To provide some understanding the space of positive determinant matrices, we provide several figures which will show the nonconvexity of the space for 2×2 and 3×3 matrices. These matrices are the representation of linear computations for 2 and 3 node systems respectively. A connected 2 node system forms a complete graph and is therefore not an example of a system performing distributed computation since both nodes have full state information. We consider this system as it is the easiest to represent graphically. Consider first a matrix

$$M_2 = \begin{bmatrix} a & b \\ c & t \end{bmatrix}, \tag{49}$$

where $M \in \mathbb{R}^{2\times 2}$. In Figure 4, the matrix M_2 is plotted with various values for its entries. Even for the smallest possible 2 node system, the space of positive determinant matrices is non-convex. By adding only one more agent, the state transition matrix has 9 entries. The

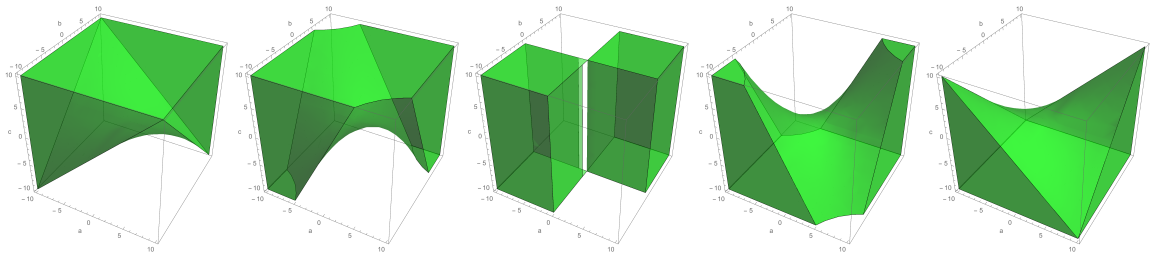
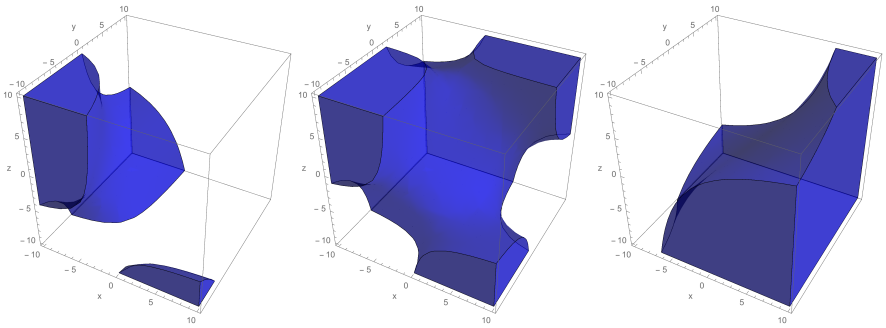


Figure 4. The shaded green region in the plots above show values for the entries in the matrix M, the axes in the volume plot represent various values for the entries a, b, and c. The plots from right to left are of different values for the entry of the matrix t. These values are t=-10,-3,0,3,10 from left to right.

3 node system coupled with a line graph as the information exchange graph is the simplest nontrivial system that can perform distributed computation. This new transformation matrix M_3 seems even less comprehensible when attempting to visualize the set of positive determinant 3×3 matrices. This is illustrated in figure 5.



$$\begin{bmatrix} x & 10 & -4.7 \\ 1.9 & y & 10 \\ -2.8 & 5.4 & z \end{bmatrix} \qquad \begin{bmatrix} a & 10 & -8.2 \\ 1.4 & e & -2.5 \\ 2.5 & -10 & i \end{bmatrix} \qquad \begin{bmatrix} x & 10 & 3.5 \\ 10 & y & 10 \\ 4.7 & -10 & z \end{bmatrix}$$

Figure 5. Here are several plots of positive determinant matrices where x, y, z are allowed to vary between -10 and 10. The rest of the matrix entries are fixed. The resulting shaded region provides the set of matrices with positive determinant. With a 3×3 matrix, non-convex disconnected sets can be generated for these particular parameter sets. It is important to note that $GL^n_+(\mathbb{R})$ is connected. Disconnected sets can be formed from $GL^3_+(\mathbb{R})$ by fixing 6 of the entries in the 3×3 matrix, even though the full 9 dimensional set is connected. Beyond 4 dimensions it becomes very hard to have geometric intuition about these sets.

Despite the lack of geometric intuition provided by visualizing these spaces, in chapter 4 a computationally fast method for planning a path between two positive determinant matrices is provided. The purpose of this section was to illustrate that the nonconvexity of

$GL^n_+(\mathbb{R})$ is a major factor which makes finding solutions to Problem 1 difficult.

3.2 A Differential Geometric Perspective on Control Difficulty

In order for a solution to Problem 1 to exist we previously showed in [37] that the desired transformation T must be in the set of $n \times n$ positive determinant matrices. This set constitutes a group together with the standard matrix product. Here this group is denoted $GL^n_+(\mathbb{R})$. This condition turns out to be both necessary and sufficient and can be stated succinctly as follows: A solution to Problem 1 exists if and only if $T \in \mathrm{GL}^n_+(\mathbb{R})$. This result can be interpreted to mean that whenever $\det(T) > 0$, there is a $W(t) \in \text{sparse}(\mathcal{G})$ that drives $\mathbf{X}$ from I to T.

Just because we know that a transformation T can be performed using local rules it does not follow that we can (easily) find these rules, encoded through $W(t) \in \text{sparse}(\mathcal{G})$, such that $\dot{\mathbf{X}} = W\mathbf{X},\ \mathbf{X}(t_0) = I, \mathbf{X}(t_f) = T$. In order to better understand how difficult finding these weights are for particular transformations and graph topologies, the relationship between graph structure and degree of nonholonomy of the drift free dynamics are explored. Each particular choice of connected information exchange graph, $\mathcal{G}$, has a different edge set and requires different wighted interaction rules in order to compute a target transform. In order to develop these results, it is most convienent to phrase the dynamical system $\dot{\mathbf{X}} = W\mathbf{X}$ in vectorized drift free form. Recall the structure of this system is,

$$\mathrm{vec}(\dot{\mathbf{X}}) = \sum_{i=1}^{n} \sum_{j|(i,j)\in E} \vec{g}_{ij}(\mathbf{X}) w_{ij}, \quad \vec{g}_{ij}(\mathbf{X}) = \mathrm{vec}(\mathbb{I}_{ij}\mathbf{X}). \tag{50}$$

In order to establish the relationship between the degree of nonholonomy of this system and the underlying communication graph we start by recalling Lemma 2, which provides an expression for computing a lie bracket,

$$[\vec{g}_{ij}(\mathbf{X}), \vec{g}_{kl}(\mathbf{X})] = \begin{cases} -\vec{g}_{il}(\mathbf{X}) & \text{if } j = k,\ i \neq l \\ \vec{g}_{kj}(\mathbf{X}) & \text{if } i = l,\ j \neq k \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{51}$$

To better understand what Lemma 2 really states, note that for every edge $(i, j) \in E$ there exists a corresponding vector field $\vec{g}_{ij}(\mathbf{X})$. One way to interpret the Lie bracketing operation that follows from Lemma 2 is that when applied to vector fields corresponding to adjacent edges, it creates new vector field corresponding to a new edge.

For example, consider the directed three node line graph shown in Figure 6. The directed edges (i, j) and (j, k) each have an associated vector field, $\vec{g}_{ij}(\mathbf{X})$ and $\vec{g}_{jk}(\mathbf{X})$ respectively. Applying Lemma 2, the Lie bracket is $[\vec{g}_{jk}(\mathbf{X}), \vec{g}_{ij}(\mathbf{X})] = \vec{g}_{ik}(\mathbf{X})$. This resulting vector field is associated with adding the edge (i, k) to the graph. So, by modulating vector fields associated with edges in a particular information-exchange graph we can make the graph behave as if it has additional edges. Consider the nested Lie bracket operation

$$[\vec{g}_{kl}(\mathbf{X}), [\vec{g}_{jk}(\mathbf{X}), \vec{g}_{ij}(\mathbf{X})]] \tag{52}$$

which can be simplified to

$$[\vec{g}_{kl}(\mathbf{X}), \vec{g}_{ik}(\mathbf{X})] = \vec{g}_{il}(\mathbf{X}) \tag{53}$$

This application shows that by nesting Lie brackets, virtual edges connecting nodes of increasing distances away can be created. In this example, a vector field associated with an edge between nodes i and l was created. These nodes were separated with a distance of 3 and were virtually connected with two nested lie brackets. *In general, through repeated applications of Lemma 2, any connected graph can be made to behave like a complete graph.*

Now, that a Lie bracketing relationship has been established for the drift free system, several definitions are needed in order to connect Lie brackets to the properties of the information exchange graph $\mathcal{G}$.

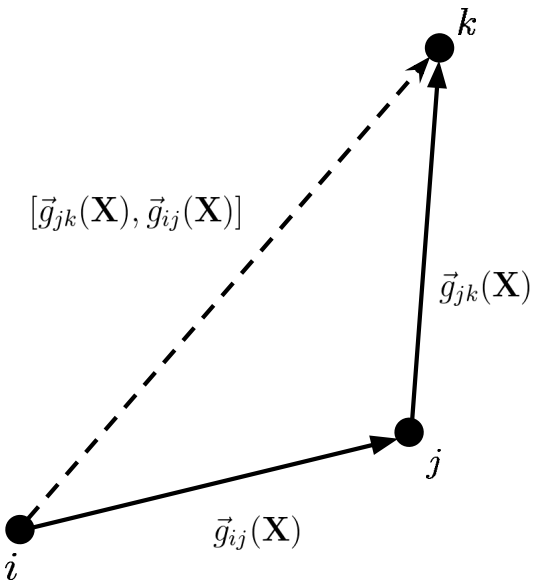


Figure 6. For every edge in the information exchange graph $\mathcal{G}$ connecting nodes i and j, there is a corresponding vector field $\vec{g}_{ij}(\mathbf{X})$. The Lie bracket operation when applied to the vector fields corresponding two two adjacent edges is equal to a vector field, $\vec{g}_{ik}(\mathbf{X})$, associated with an edge connecting node i to node k.

Definition 1 (Distance)**.** *The distance between two nodes in a graph u, v is denoted $d(u, v)$. The value of $d(u, v)$ is the cardinality of the edge set that comprises a shortest path connecting the two nodes.*

Definition 2 (Graph Diameter)**.** *The* Graph Diameter *D of a graph $\mathcal{G} = (V, E)$ is* $\max_{u,v} d(u, v)$ *for any $u, v \in V$.*

In order to talk about the degree of nonholonomy of a drift free system, the notion of a distribution and filtration must be introduced. $\Delta(x)$, a distribution of a drift free system as in (10), is defined as

$$\Delta(x) = \text{span}\{g_1(x), \ldots, g_q(x)\}. \tag{54}$$

The concept of filtration is also needed. A filtration is defined as follows: $G_i = G_{i-1} + [G_1, G_{i-1}]$ where $G_1 = \Delta$ and $[G_1, G_{i-1}] = span\{[g, h] | g \in G_1, h \in G_{i-1}\}$. If a filtration does

not increase the dimension of the space spanned by G_i then that distribution is said to be *involutive*.

Definition 3 (Degree of Nonholonomy)**.** *The degree of nonholonomy is given by the least number of filtrations p required to reach an involutive distribution. (See for example [39].)*

Using the above definitions, the following lemma is presented in order to relate a particular vector field to the number of filtration required to form it.

Lemma 6. *For any* $u, v \in V$, $\vec{g}_{uv}(\mathbf{X}) \in G_{d(u,v)-1}$.

Proof. Because $\mathcal{G}$ is strongly connected, every node pair $u, v \in V$ is path-connected. Path-connected means that there is a path through adjacent nodes in the graph $\mathcal{G}$ that starts at node u and ends at node v. The distance between the two nodes, $d(u, v)$ gives the number of edges in the shortest path connecting u and v. Using that fact, assume that the path goes through the nodes $N_1, \ldots, N_{d(u,v)+1}$, i.e., N_1 is adjacent to N_2, N_2 is adjacent to N_3, and so forth, while $N_1 = u$ and $N_{d(u,v)+1} = v$. Since these nodes are adjacent, we, by definition, have that $\vec{g}_{N_1N_2}, \vec{g}_{N_2N_3}, \ldots, \vec{g}_{N_{d(u,v)}N_{d(u,v)+1}} \in \Delta(\mathbf{X})$.

Since the involutive closure contains every possible Lie bracket that can be recursively created from elements $\Delta(\mathbf{X})$, the problem is to create $\vec{g}_{uv}$ from some combination of Lie brackets from elements in $\Delta(\mathbf{X})$. And, from Lemma 2, we know that $[\vec{g}_{N_2N_3}, \vec{g}_{N_1N_2}]$ is equal to $\vec{g}_{N_1N_3}$. Applying Lemma 2 again gives $[\vec{g}_{N_3N_4}, \vec{g}_{N_1N_3}] = \vec{g}_{N_1N_4}$. If this procedure is recursively applied $d(u, v) - 1$ times, we arrive at $[\vec{g}_{N_{d(u,v)}N_{d(u,v)+1}}, \vec{g}_{N_1N_{d(u,v)}}] = \vec{g}_{N_1N_{d(u,v)+1}}$. So, we are able to construct $\vec{g}_{N_1N_{d(u,v)+1}}$ from previous Lie brackets. And, as $N_1 = u$ and $N_{d(u,v)+1} = v$, $d(u, v) - 1$ lie bracketing operations are required and therefore $d(u, v) - 1$ filtrations are required. So, we arrive at $\vec{g}_{uv} \in G_{d(u,v)}$. □

The previous result can be used to relate the distance between any two nodes in the computation graph to the degree of nonholonomy of the system. This theorem can be intuitively understood by noting that the number of filtrations needed to form a particular

vector field is equal to the distance between a pair of nodes. To link the information-exchange graph to the degree of nonholonomy of (15), one final intermediate result is needed. Using the lemmas developed above we can prove the main result of this paper:

Theorem 4. *The Degree of Nonholonomy of (15) is equal to* $D-1$ *where* D *is the diameter of the computation graph* $\mathcal{G}$*.*

Proof. By definition the diameter of $\mathcal{G}$ is $D = \max_{u,v} d(u,v)$. Because, $G_i = G_{i-1} + [G_1, G_{i-1}]$, for any $i < D-1$, $G_i \subset G_{D-1}$. Since D is the maximum distance of any path between two nodes in $\mathcal{G}$, it follows from Lemma 6 that G_{D-1} is the smallest possible filtration that contains every $\vec{g}_{uv}(\mathbf{X})$, i.e., for any $u, v \in V$ $\vec{g}_{uv}(\mathbf{X}) \in G_{D-1}$, because there is at least one pair of nodes u, v which has a shortest path of length D. By Lemma 4 this set of vectors is linearly independent. Since we know that there are n^2 such vectors it can be concluded that they span the space of $\mathbb{R}^{n^2}$. Therefore, G_{D-1} spans the entire state space of (15) and is involutive. It follows that $D-1$ is the degree of nonholonomy of (15). □

Theorem 4 provides a connection between graph structure and dynamic constraints. The Lie bracket operation on adjacent edges can be interpreted as information flow along edges of a system. If information is to flow between node i and j, the system can travel along the vector field $\vec{g}_{ij}(\mathbf{X})$. Equivalently, information flow along $\vec{g}_{jk}(\mathbf{X})$ occurs when information flows between node j and k. The Lie bracket corresponding to $[\vec{g}_{jk}(\mathbf{X}), \vec{g}_{il}(\mathbf{X})]$ creates a vector field corresponding to information flow between nodes i and k. In this way, the result in Theorem 4 becomes intuitive. In order to have information flowing between every node in the graph for a global computation, there must be a vector field corresponding to information flow between each pair of nodes. Considering that the diameter of a graph is the maximum path length between any two nodes, the number of informational hops required to connect each node to every other node is equal to $D-1$. Additionally, the Lie bracket operation increases in order for each required hop. So, information flow and the Lie bracket operation are inherently tied together.

Theorem 4 also has implications on the synthesis of controllers. If the interaction exchange graph is complete, then the system is holonomic and a weight matrix can be trivially synthesized. However, if even one edge is missing in the information exchange graph the degree of nonholonomy is nonzero. From [42] we know that there exists no smooth feedback control law for nonholonomic systems. In general, nonholonomic systems are regarded as harder to control. As such, we propose numerical methods for finding the weight based interaction rules which compute a transformation of interest.

3.3 Brockett's Theorem and Its Implications

One implication of Brockett's Theorem stated in [42] is for drift free systems of the form,

$$\dot{x} = \sum_{i=1}^{m} g_i(x)u_i \tag{55}$$

with $x \in \mathbb{R}^n$, this system can only be stabilized with smooth feedback if $m = n$. This condition only holds in the case of the information-exchange graph being the complete graph. Or in other words, the system must be holonomic. This relationship is formalized in the previous section by Theorem 4. This means that we can find smooth feedback controllers to stabilize to a given target transformation if and only if $\mathcal{G}$ is the complete graph. If even one edge is removed from the complete graph, there is no longer a smooth feedback controller which provides weights for the computation of a target transformation T.

In order to illustrate this concept, a control Lyoponov function approach is developed. Then we attempt to synthesize weighted interaction rules to compute a target transformation T under two different topologies, a complete graph ($\mathcal{G}_c$), and a complete graph with one pair of edges removed ($\mathcal{G}_s$). This will illustrate the difficulty that sparsity introduces into finding weighted interaction rules.

To this end, suppose we are trying to control a system as in Problem 1,

$$\dot{\mathbf{X}} = W\mathbf{X} \tag{56}$$

$$W \in \text{sparse}(\mathcal{G}) \tag{57}$$

where at time t_f, we desire to reach state $\mathbf{X}(t_f) = T$. One approach to this problem would be to define a dynamical constraint which would eventually stabilize to our target matrix. In order to approach this problem, we choose to define a Lyoponov function and force our choice of weighting functions to cause asymptotic convergence. Let the Lyoponov function be,

$$V(\mathbf{X}) = \frac{1}{2}||\mathbf{X} - T||_F^2 \tag{58}$$

Now taking the first time derivative yields

$$\dot{V}(\mathbf{X}) = \text{Tr}((\mathbf{X} - T)^T \dot{\mathbf{X}}) = \text{Tr}(\mathbf{X}(\mathbf{X} - T)^T W) \tag{59}$$

For asymptotic stability Lyoponov tells us:

$$\dot{V}(\mathbf{X}) < 0 \implies \text{Tr}(\mathbf{X}(\mathbf{X} - T)^T W) < 0 \tag{60}$$

This condition can be phrased as a constraint that is linear in the elements of W. let the nonzero elements of the weight matrix W be gathered into a vector expressed as $\mathbf{w}$. This condition can be stated as

$$\text{vec}((\mathbf{X} - T)\mathbf{X}^T)G\mathbf{w} < 0 \tag{61}$$

where G is an $n \times m$ matrix which preserves the sparsity condition on W. We satisfy this condition and force exponential convergence by having a more strict bound on the decrease of the Lyoponov Function with time. To do this restrict the constraint to

$$\text{vec}((\mathbf{X} - T)\mathbf{X}^T)G\mathbf{w} = -k||\mathbf{X} - T||_F \tag{62}$$

For simplicity of notation let $\text{vec}((\mathbf{X} - T)\mathbf{X}^T)G$ be represented by $C(\mathbf{X})$. Once we have forced the constraint to take a particular value we can explicitly solve for the weight vector $\mathbf{w}$. This is done as follows

$$C(\mathbf{X})\mathbf{w} = -||\mathbf{X} - T||_F \tag{63}$$

$$C(\mathbf{X})^T C(\mathbf{X})\mathbf{w} = -C(\mathbf{X})^T||\mathbf{X} - T||_F \tag{64}$$

$$\mathbf{w} = -(C(\mathbf{X})^T C(\mathbf{X}))^{-1} C(\mathbf{X})^T||\mathbf{X} - T||_F \tag{65}$$

Now this control law can be used as a feedback control law that should stabilize the state to the target transformation T. In order to test this approach, consider a 4 node system. We attempt to synthesize weighted interaction rules for the transformation,

$$T = \begin{bmatrix} -6 & 4 & 6 & -3 \\ 0 & -2 & -3 & 5 \\ 10 & -10 & -9 & 1 \\ 0 & -4 & 0 & 6 \end{bmatrix} \tag{66}$$

using the control Lyoponov function approach outlined above for both a complete interaction-exchange graph ($\mathcal{G}_c$) and a complete graph with one pair of edges removed ($\mathcal{G}_s$). The results of this are shown in Figure 7.

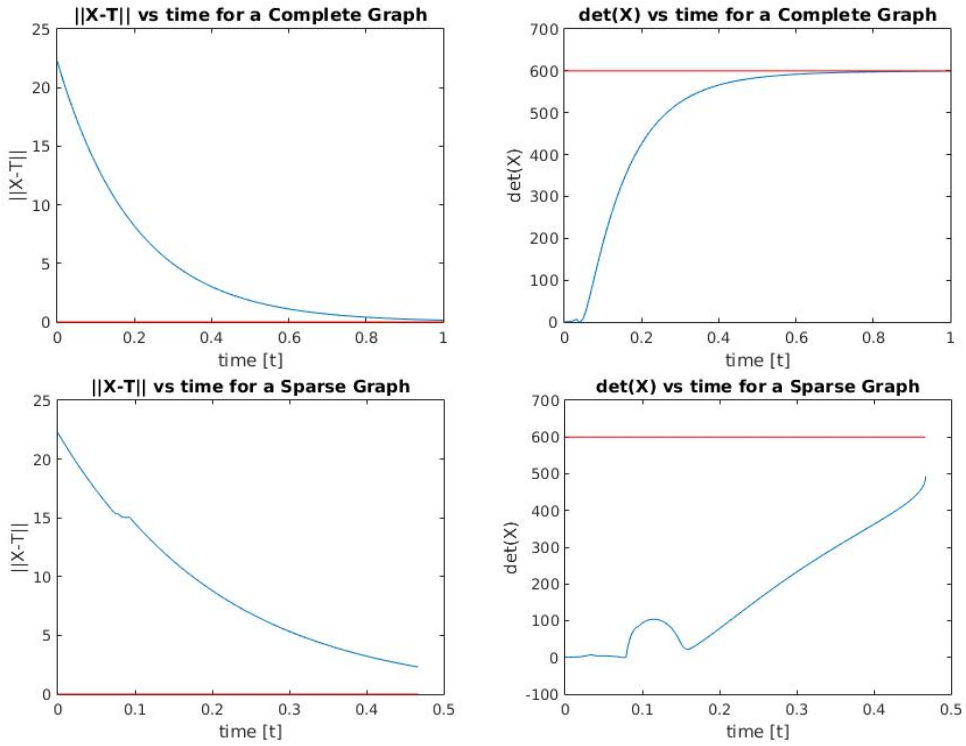


Figure 7. The above plots compare convergence of a control lyoponov function approach to finding weights which compute a target transform T. In this case we compare a 4 node system with a complete information exchange graph (upper plots) against an information exchange graph with one edge pair missing. The upper plots show the evolution of a holonomic system. The plots on the bottom show a nonholonomic system. The nonholonomic system stalls out short of reaching convergence to the target transformation whereas the holonomic system smoothly converges. These results are in line with what should be expected from Brockett's Theorem.

In this section we have illustrated that smooth feedback controllers can not be used to synthesize weighted interaction rules. As a result, an optimal control approach is developed in the next chapter which results in a control policy parameterized by time.

CHAPTER 4

COMPUTING SOLUTIONS

In the previous chapter, the difficulty of finding solutions to Problem 1 were discussed in terms of interaction exchange graph topology. Brockett's Theorem states that there is no smooth feedback control which stabilizes a nonholonomic drift free system to a target point. So as a result we pursue an optimal control based method to numerically find time varying weighting functions. This chapter is devoted to developing two different optimal control problem formulations and numerically solving them for several example test cases. Both a shooting method and Pseudospectral method are used and results are presented.

4.1 Optimal Control Problem Formulations

As Brockett's Theorem says that no smooth feedback control can stabilize a system, we must instead look for time varying controls to drive the system from the initial condition $\mathbf{X}(t_0) = I$ to the final target transformation $\mathbf{X}(t_f) = T$. One technique that can be used to find these time varying weighting functions is optimal control. This involves optimizing a cost functional over a finite time horizon. Ultimately this problem reduces to solving a nonlinear two-point boundary value problem. In order to synthesize efficient interaction rules, we formulate two optimal control problems, where the inputs are defined as the time-varying weights.

Just because we know that a computation $T\xi$ can be done using local rules it does not follow that we can (easily) find these rules, encoded through $W(t) \in \text{sparse}(G)$, such that $\dot{\mathbf{X}} = W\mathbf{X},\ \mathbf{X}(t_0) = I, \mathbf{X}(t_f) = T$. There are many possible ways in which weight functions can be found. In this chapter, we address this problem of finding weighting functions $W(t)$ which allow for the computation of a provided target transformation T. We considered two cost functions. One which minimizes the energy of the weighting functions, and one which tries to track a path through $GL^n_+(\mathbb{R})$.

4.1.1 Energy Minimization Problem

There are many possible ways in which weight functions can be found. As such, when framed as an optimal control problem, many costs can be considered. If the goal is to simply find a set of weights which reaches some desired transformation, any choice of cost is valid. In this case, we have chosen to minimize the energy of the weight functions. So, we consider the cost,

$$J(W) = \int_0^{t_f} \frac{1}{2} \|W(t)\|_F^2 dt, \tag{67}$$

where $\|\cdot\|_F$ is the Frobenius norm. From this cost we can create a minimization problem which when solved will provide local interaction rules to compute a desired linear transformation on the system states. The resulting constrained minimization problem becomes

Problem 3 (Optimal Local Interactions)**.**

$$\min_W J(W) = \int_{t_0}^{t_f} \frac{1}{2} \|W(t)\|_F^2 \, dt$$

subject to the constraints

$$\dot{\mathbf{X}} = W\mathbf{X} \quad W(t) \in sparse(G)$$

$$\mathbf{X}(t_0) = I \qquad \mathbf{X}(t_f) = T$$

Under this problem formulation, only the start and end points are constrained while the weight magnitudes are penalized. We also consider the case where following a particular path is important.

4.1.2 A Tracking Problem

Instead of considering point constraints, we explore the case where following a chosen path is important. In order to synthesize a control which will guide the dynamical system in

Problem 1, we first find a feasible path from the identity matrix to the target transformation $T \in \mathbb{R}^{n\times n}$. According to [37] and [43], all reachable transformations are in the group $GL_n^+(\mathbb{R})$. That is to say, any target transformation T must have positive determinant. We proceed by providing a general method to generate a path between the identity matrix and a given target transformation T where every intermediate point on the path has positive determinant.

Given a non-singular matrix M, it can always be written as

$$M = KP, \tag{68}$$

with $K \in O(n)$ where $O(n)$ is the orthogonal group, the set of all matrices $K \in \mathbb{R}^{n\times n}$ where $det(K) = \pm 1$ and $P = P^T \succ 0$. If $M \in GL_n^+(\mathbb{R})$ then

$$M = RP, \tag{69}$$

where R is a rotation matrix, i.e., $R \in SO(n)$ [44].

So, given a target transformation T that we hope to achieve at time $t = t_f$, we can plan paths through $GL_n^+(\mathbb{R})$ using the polar decomposition. The target can be decomposed as

$$T = R_T P_T. \tag{70}$$

Now, since we are moving from $\mathbf{X}(t_0) = I$ to $\mathbf{X}(t_f) = T$, we can simply plan a path between these two matrices using a polar decomposition. Let $M(t) = R(t)P(t)$ be the planned path with

$$R(0) = I \quad P(0) = I \quad R(t_f) = R_T \quad P(t_f) = P_T \tag{71}$$

Additionally, let $\phi(t)$ be the mapping from the interval $[t_0, t_f]$ onto the interval $[0, 1]$. The geodesic from I to P_T in the space of positive definite matrices (in Frobenius norm) [45] is

$$P(t) = P_T^{\phi(t)}. \tag{72}$$

The path through $SO(n)$ can be obtained using the idea in [46] of using constant angular velocity

$$\dot{R} = \hat{\Omega}R, \tag{73}$$

where $\hat{\Omega} \in so(n)$ is skew-symmetric, with the result that

$$R(t) = e^{\hat{\Omega}\phi(t)}. \tag{74}$$

Moreover, $\hat{\Omega}$ is simply given by the skew symmetric and real logarithm

$$\hat{\Omega} = ln(R_T), \tag{75}$$

which gives the final path as

$$M(t) = e^{ln(R_T)\phi(t)} P_T^{\phi(t)}. \tag{76}$$

Using the above derivation we can formulate an optimal control problem.

Problem 4 (Optimal Tracking)**.**

$$\min_{W(t)} \int_{t_0}^{t_f} \|\mathbf{X}(t) - M(t)\|_F^2 \, dt$$

subject to the constraints

$$\dot{\mathbf{X}} = W\mathbf{X} \quad W(t) \in sparse(G)$$

$$\mathbf{X}(t_0) = I \qquad\qquad \mathbf{X}(t_f) = T$$

This second problem when solved tracks the designated path $M(t)$. In the section that follows both Problem 3 and 4. are simulated and the differences in the synthesized state trajectories are explored.

4.2 Deriving the Two Point Boundary Value Problem for Shooting

In order to use a test shooting method to solve Problem 3, first the associated two point boundary value problem must be derived. To do this we form the Hamiltonian. The Hamiltonian associated with Problem 3 (e.g., [47]), with costate matrix λ, is given by

$$H = \text{vec}(\lambda)^T \text{vec}(W\mathbf{X}) + \frac{1}{2}\|W\|_F^2. \tag{77}$$

We can rewrite the Hamiltonian as

$$H = \sum_{i=1}^{n} \sum_{j|(i,j)\in E} \sum_{k=1}^{n} \lambda_{ik} w_{ij} \mathbf{X}_{jk} + \frac{1}{2} \sum_{i=1}^{n} \sum_{j|(i,j)\in E} w_{ij}^2. \tag{78}$$

The optimality conditions are

$$0 = \frac{\partial H}{\partial w_{ij}} = \sum_{k=1}^{n} \lambda_{ik} \mathbf{X}_{jk} + w_{ij}, \tag{79}$$

i.e., the optimal weights are given by

$$w_{ij} = -\sum_{k=1}^{n} \lambda_{ik} \mathbf{X}_{jk}, \tag{80}$$

which yields $m + n$ optimality conditions. This is also the number of nonzero values in the W matrix. We get the costate equations from the derivative of the Hamiltonian with respect to $\mathbf{X}$:

$$\dot{\lambda}_{ij} = -\frac{\partial H}{\partial \mathbf{X}_{ij}} = -\sum_{k|(i,k)\in E} w_{ki} \lambda_{kj}. \tag{81}$$

By substituting the optimality conditions into both the state and costate equations, we get $2n$ equations with initial and final conditions on $\mathbf{X}_{ij}$. The resulting, two-point boundary problem becomes

$$\begin{aligned} \dot{\mathbf{X}}_{ij} &= -\sum_{k|(i,k)\in E} \mathbf{X}_{kj} \sum_{l=1}^{n} \lambda_{il} \mathbf{X}_{kl} \\ \mathbf{X}(t_0) &= I,\ \mathbf{X}(t_f) = T \\ \dot{\lambda}_{ij} &= \sum_{k|(i,k)\in E} \lambda_{kj} \sum_{l=1}^{n} \lambda_{kl} \mathbf{X}_{il}, \end{aligned} \tag{82}$$

This system can be succintly rewritten in ensamble form as,

$$\begin{aligned} \dot{\mathbf{X}} &= -(\hat{G} \odot \lambda \mathbf{X}^T)\mathbf{X} \\ \mathbf{X}(t_0) &= I,\ \mathbf{X}(t_f) = T \\ \dot{\lambda} &= (\hat{G} \odot \mathbf{X} \lambda^T)\lambda \end{aligned} \tag{83}$$

where $\lambda \in \mathbb{R}^{n\times n}$ is a matrix whose entries are costates and $\hat{G} \in \text{sparse}(\mathcal{G})$ is a matrix whose nonzero entries are equal to 1. The numerical solution for the weight functions were found

in the example by solving (83) using test shooting. A reference which explains this method in detail is [48]. Using optimal control to find weight functions was simply a convenient method for illustrating the feasibility of finding solutions.

4.2.1 A Numerical Example Using Test Shooting

Consider the situation when the linear transformation T represents a reordering (or swapping) of states. We examine the 4 node case where the underlying graph topology is given by nodes $2, 3, 4$ forming a clique (fully connected subgraph) and node 1 is connected to node 2. In this example agents 1 and 2 and agents 3 and 4 are to "swap" state values, the transformation matrix becomes

$$T_{swap} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{84}$$

However, the linear interpolation between I and T_{swap} contains a singular matrix, which makes the two-point boundary problem numerically ill-conditioned when using shooting methods, e.g., [48]. There are many such choices of transformations where this ill-conditioning is a concern, as discussed in [49]. A way around this problem is to avoid this singular matrix by solving two sequential two-point boundary problems.

As an example, in the first iteration, we let the boundary conditions be $\mathbf{X}(t_0) = I,\ \mathbf{X}((t_f - t_0)/2) = T_1$. For the second iteration, they are $\mathbf{X}((t_f - t_0)/2) = T_1,\ \mathbf{X}(t_f) = T_{swap}$, where

$$T_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{85}$$

This sequential approach avoids the numerical ill-conditioning, and the solution is shown in Figures 8 - 9. This method works well for small systems where the number

of agents is 5 or less. As systems get larger the amount of computation required to find a solution becomes very large. As a result, we explored a second pseudospectral method for solving the optimal control problems posed.

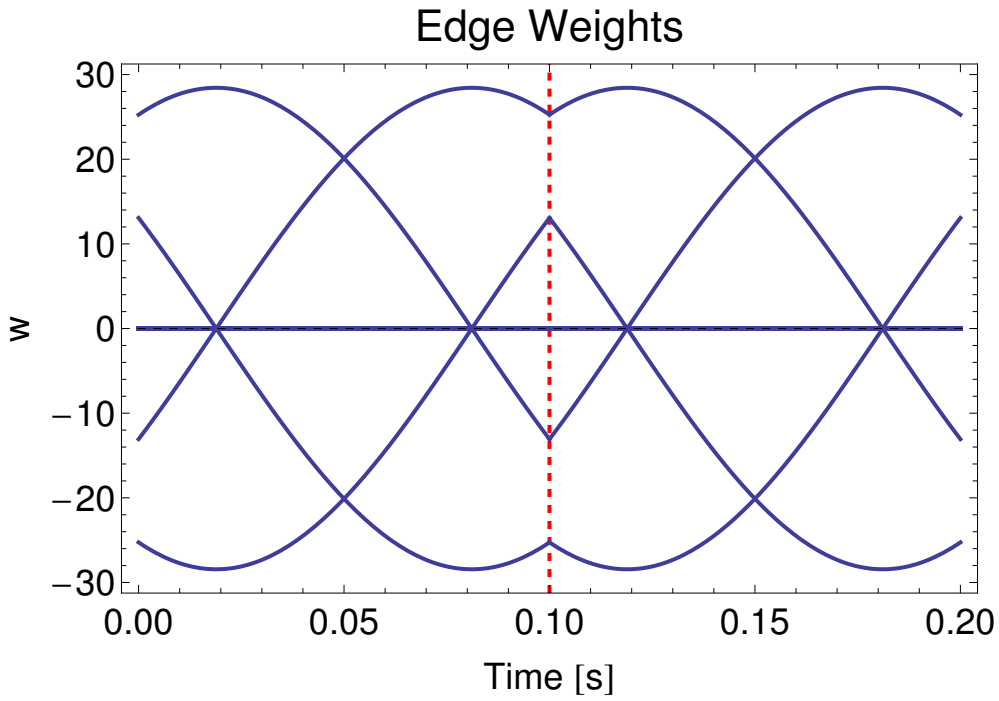


Figure 8. The weight functions define the local interactions needed to achieve the swap in the 4-node case. The first and second subproblems are solved over the time intervals $[0, 0.1)$ **and** $[0.1, 0.2]$ **respectively.**

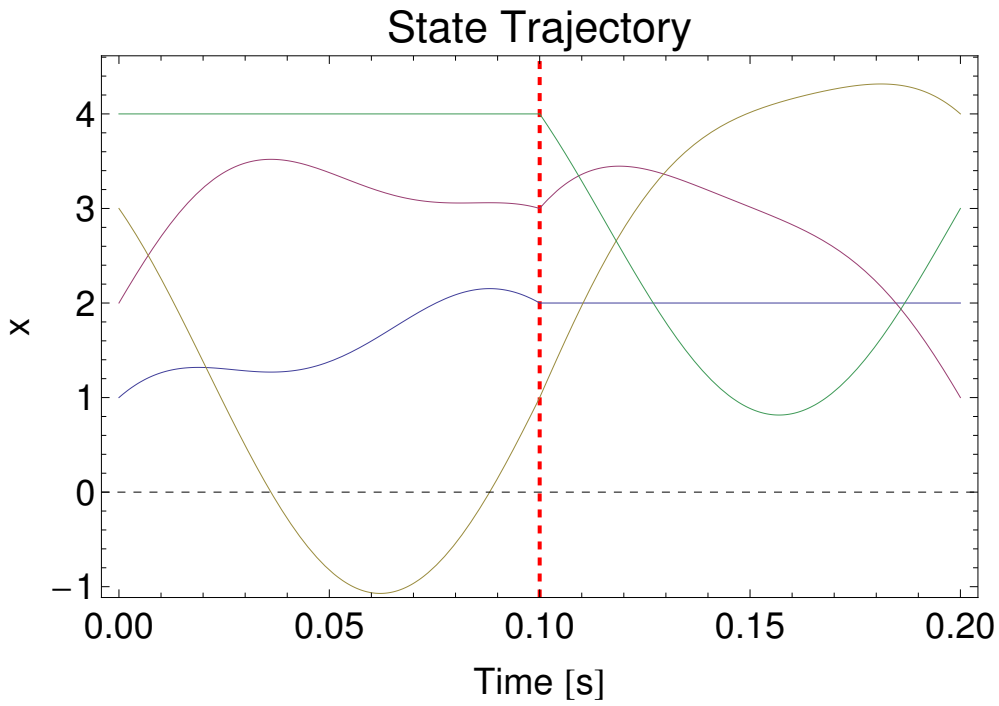


Figure 9. The evolution of the node states for the swap problem. The initial state is $x(t_0) = [1, 2, 3, 4]^T$ and the final state is $x(t_f) = [2, 1, 4, 3]^T$, i.e., the first and second states swapped values and the third and fourth state swapped values.

4.3 The Pseudospectral Method

Existing analytic approaches do not scale well for such medium-to-large scale nonlinear problems. In this work we find solutions using a direct collocation computational method, which discretizes the continuous-time optimal control problem into a nonlinear optimization problem whose decision variables are the values of the discretized state trajectories and control signals. We show this combined approach is general and an efficient approach to deriving the weights needed for local computations.

The pseudospectral method was originally developed in order to solve partial differential equations but has been applied more widely to optimal control problems in a variety of applications such as satellite maneuvers, quantum control, and neuroscience [50, 51, 52]. Like many methods, this approach discretizes the continuous optimal control problem into a nonlinear programing problem for which there are many commercial and open-source

solvers available. The method relies on a connection between two families of polynomials. The orthogonal Legendre polynomials represent the "spectral" portion of the method, providing a level of exponential convergence as the order of the discretization increases. The Lagrange polynomials composes the "psuedo" part of the method and makes the method significantly easier to implement by dealing directly with interpolated state and control values as decision variables rather than the expansion coefficients of the Legendre polynomials, which have less physical interpretation. With this method we can consider an optimal control problem of Bolza form,

$$\begin{aligned}
\min_{W(t)} \ & \varphi(t_f, \mathbf{X}(t_f)) + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{X}(t), W(t))\, dt, \\
\text{s.t.} \ & \dot{\mathbf{X}}(t) = W(t)\mathbf{X}(t), \\
& e(\mathbf{X}(0), \mathbf{X}(t_f)) = 0, \\
& g(\mathbf{X}(t), W(t)) \leq 0, \quad \forall\ t \in [t_0, t_f],
\end{aligned}$$

where φ and $\mathcal{L}$ are the terminal and running costs and e and g are endpoint and path constraints, respectively. This optimal control problem is transformed to a nonlinear programming problem of the form,

$$\begin{aligned}
\min_{\bar{\mathbf{X}}, \bar{W}} \ & \varphi(\bar{\mathbf{X}}_N) + \frac{T}{2} \sum_{i=0}^{N} \mathcal{L}(\bar{\mathbf{X}}_i, \bar{W}_i) w_i, \\
\text{s.t.} \ & \sum_{k=0}^{N} D_{jk} \bar{\mathbf{X}}_k = \bar{W}_j \bar{\mathbf{X}}_j, \\
& e(\bar{\mathbf{X}}_0, \bar{\mathbf{X}}_N) = 0, \\
& g(\bar{\mathbf{X}}_j, \bar{W}_j) \leq 0, \quad \forall\ j \in \{0, 1, \ldots, N\},
\end{aligned}$$

where $\bar{\mathbf{X}} \in \mathbb{R}^{n \times n \times N}$ and $\bar{W} \in \mathbb{R}^{n \times n \times N}$ such that $\bar{\mathbf{X}}_j = \mathbf{X}(t_j)$ and $\bar{W}_j = W(t_j)$; t_j is the j^{th} interpolation point; w_i are the integration weights for quadrature approximations of integrals; and D is a constant matrix determined by the order of discretization N. More details can be found in [51]. In this work we use combination of commercial solvers, AMPL and KNITRO, to solve this nonlinear programming problem [53, 54].

4.3.1 Examples Solving Problem 3 using the Pseudospectral Method

Here we present several results illustrating the efficacy of the pseudospectral method for determining the time-varying weight functions that perform a desired global computation on robot states. First we present several simulations for particular target transformations of interest. Then we show empirical results on the computation time for various problem sizes. We chose to use a swapping computation in the examples to illustrate this method because it has obvious physical meaning for robotic systems and requires global information in order to compute the transform.

4.3.1.1 4 Robot Swap

We consider the 4 robot system where the communication architecture encoded by the structure of the matrix $W(t)$ and the desired global transformation over the time interval $t \in [0, 1]$ are given by

$$W = \begin{bmatrix} w_{11} & w_{12} & 0 & 0 \\ w_{21} & w_{22} & w_{23} & 0 \\ 0 & w_{32} & w_{33} & w_{34} \\ 0 & 0 & w_{43} & w_{44} \end{bmatrix}, \tag{86}$$

$$T_{\text{swap}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \tag{87}$$

respectively, where we have dropped the explicit dependence on t to save space, $w_{ij} = w_{ij}(t)$. T_{swap} is designed so that the final robot states are the initial states with robots 1 and 3 as well as 2 and 4 swapped, i.e., $x_1(1) = x_3(0)$, $x_4(1) = x_2(0)$, $x_3(1) = x_1(0)$, and $x_4(1) = x_2(0)$ ($x_i(t)$ is the value of the state of node j whereas $\mathbf{X}(t)$ is the transition matrix at time t). This 4 node example is simulated by solving Problem 2 using T_{swap} as the target transformation. Physically, when implemented by a system of ground robots moving in $\mathbb{R}^2$, the weights are used independently for both the x and y coordinates of the robots and the

result would be a physical swap in position between each pairs of robots, $(1, 3)$ and $(2, 4)$ regardless of their initial starting positions.

The entries of $W(t)$ and $\mathbf{X}(t)$ for the computed solution are plotted in Fig. 10, using a discretization level of $N = 15$. This solution is a realization of the proposed swap in Figure 1. This solution for local computations that lead to a global transformation is realized in hardware in the next section.

4.3.1.2 10 Robot Swap

We now consider a larger example with 10 robots and a desired transformation that swaps several of the states in a similar fashion to the previous case. A larger discretization level $N = 50$ is chosen to capture the more quickly varying inputs and trajectories required to make the higher-dimensional state transfer. The entries of $W(t)$ and $\mathbf{X}(t)$ for the computed solution are plotted in Fig. 11.

Note that any linear transformation T ($\det T \geq 0$) is possible to design in this manner. The swap matrices typically generate plots which are more easily interpreted because $T_{ij} \in \{0, 1\}$. We have likewise synthesized weights for target transformations generated randomly and achieved results with the same levels of performance and accuracy. In the case of random T matrices, the discretization value is often chosen higher, around $N = 100$.

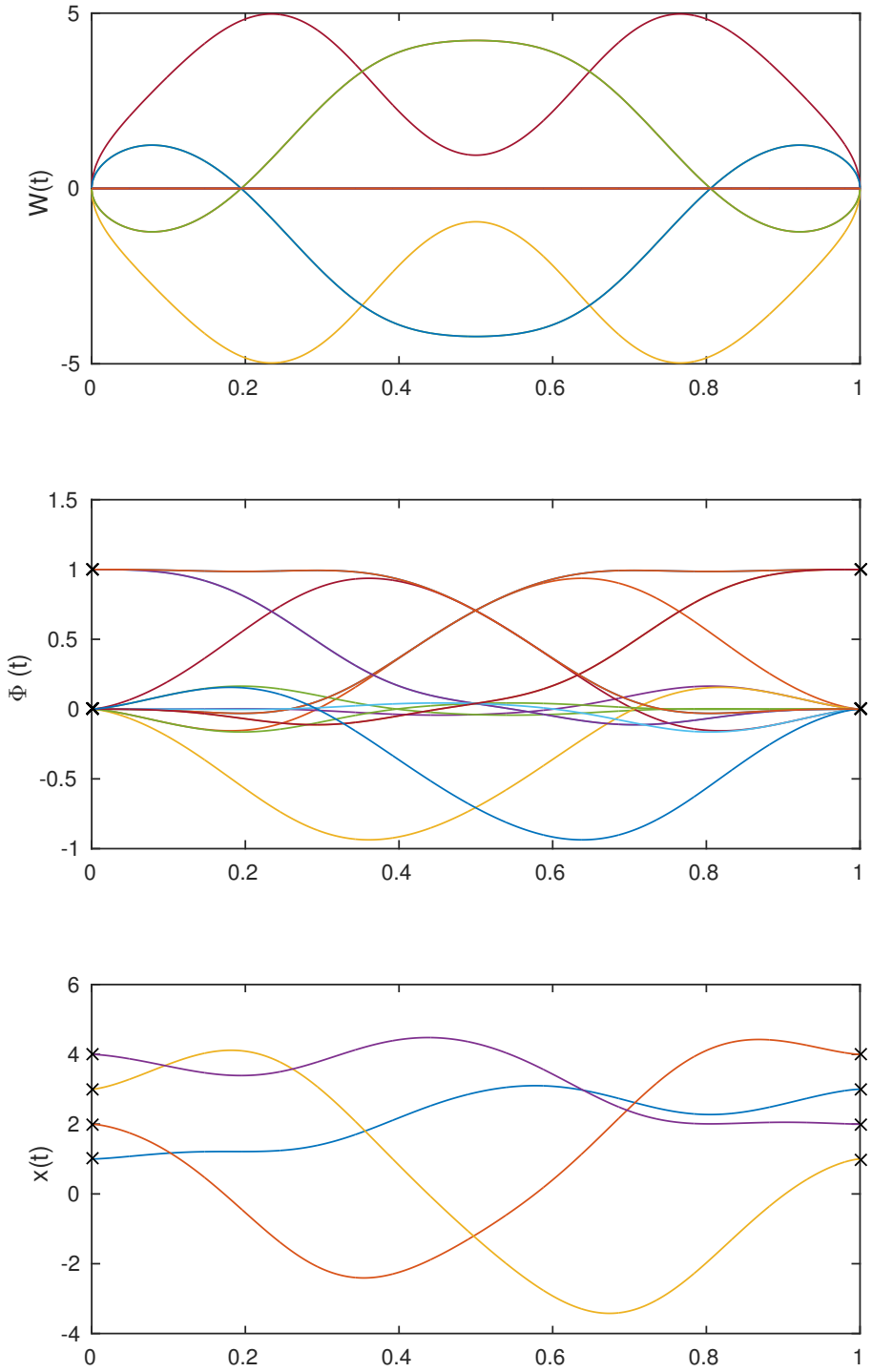


Figure 10. The time-varying weights of $W(t)$ and the trajectory of the entries of $\mathbf{X}(t)$ corresponding to a 4 node system in which the desired linear transformation swaps the values of certain nodes. In the last subfigure we show a sample evolution of node values from $x(0) = [1, 2, 3, 4]$ to $x(1) = [3, 4, 1, 2]$. The $\times$ markers indicate the entries of the desired transformation matrix T (middle) and the desired swap of state values (bottom).

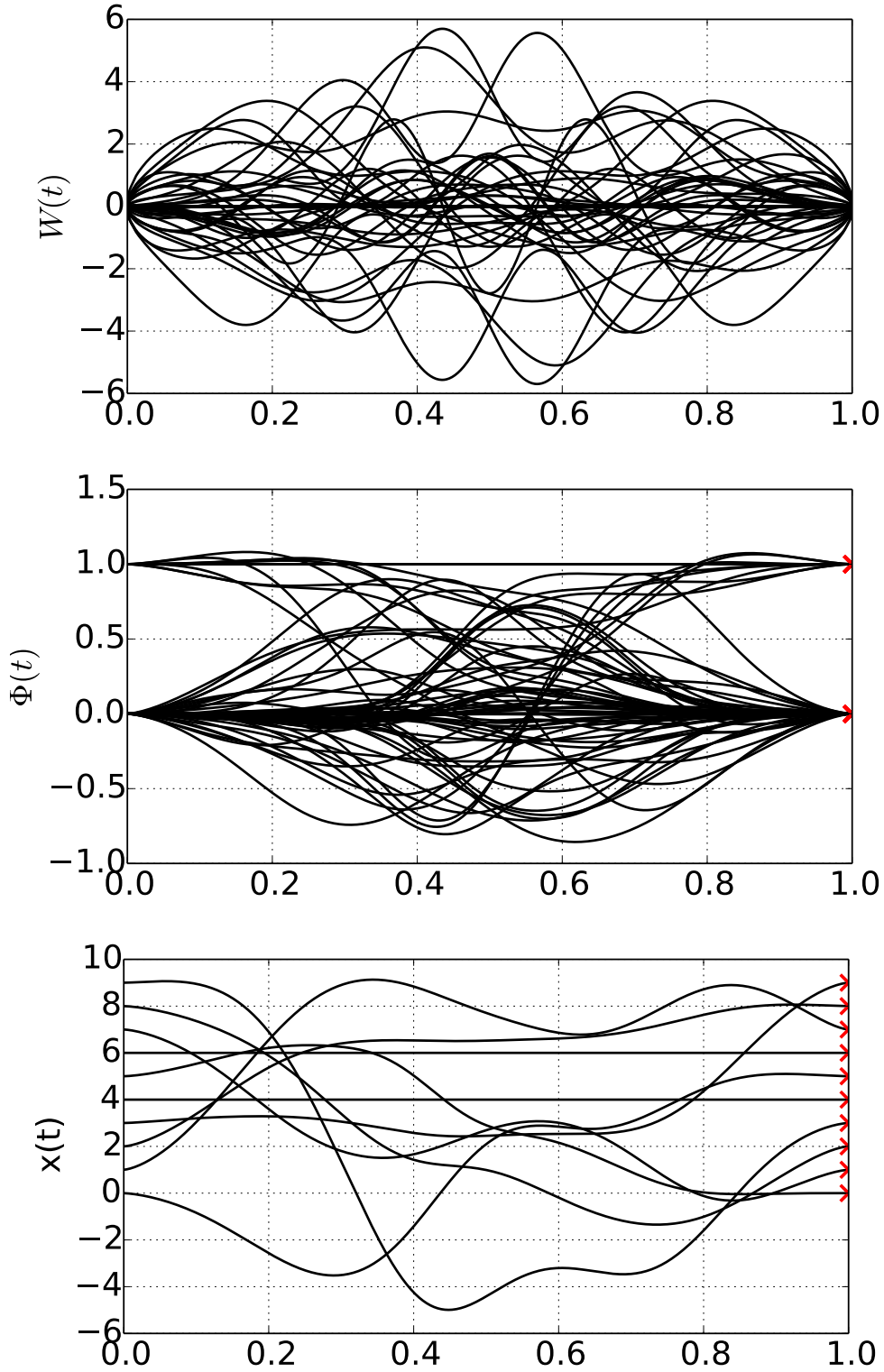


Figure 11. The time-varying weights of $W(t)$ and the trajectory of the entries of $\mathbf{X}(t)$ corresponding to a 10 node system in which the desired linear transformation swaps the values of certain nodes. In the last subfigure we show a sample evolution of node values from $x(0) = [0, \ldots, 9]$ to $x(1) = [5, 7, 8, 9, 4, 0, 6, 1, 2, 3]$. The $\times$ markers indicate the entries of the desired transformation matrix T (middle) and the desired swap of state values (bottom).

4.3.2 An Example Solving Problem 4 Using the Pseudospectral Method

We now consider the second goal of tracking a desired reference trajectory for the transition matrix $\mathbf{X}$, while still ultimately achieving the desired transfer expressed by T. Because the reference trajectory is not necessarily a feasible trajectory of the system, we cannot implement the tracking condition as a constraint. Instead is is part of the running cost term as shown in Problem 1. We include the power running cost as before with a very small relative weight to promote smooth solutions. This additional term tends to create solutions that are more numerically stable and thereby typically improves the performance of the solutions. In Fig. 12 the entries of $W(t)$ and $\mathbf{X}(t)$ for the computed solution are plotted for a discretization level of $N = 200$. The relatively rapid oscillation of this solution leads to some numerical inaccuracies, so we impose an amplitude bound $A = 10$ on the entries of $W(t)$ to guard against this, such that $|W_{ij}| \leq A,\ i, j = 1, \ldots, n$. This amplitude bound has the effect of trading off the tightness of tracking with the smoothness of the trajectories and controls, as depicted in Fig. 13 for $A = 5$ and $A = 20$. Brief violations of this amplitude bound can be seen in the plot of $W(t)$. These are due to the polynomial interpolation that produces solution functions from the solutions of the nonlinear programming problem. Such overshoots can be truncated with negligible effect on the solutions if desired, however, doing so is not critical in this context.

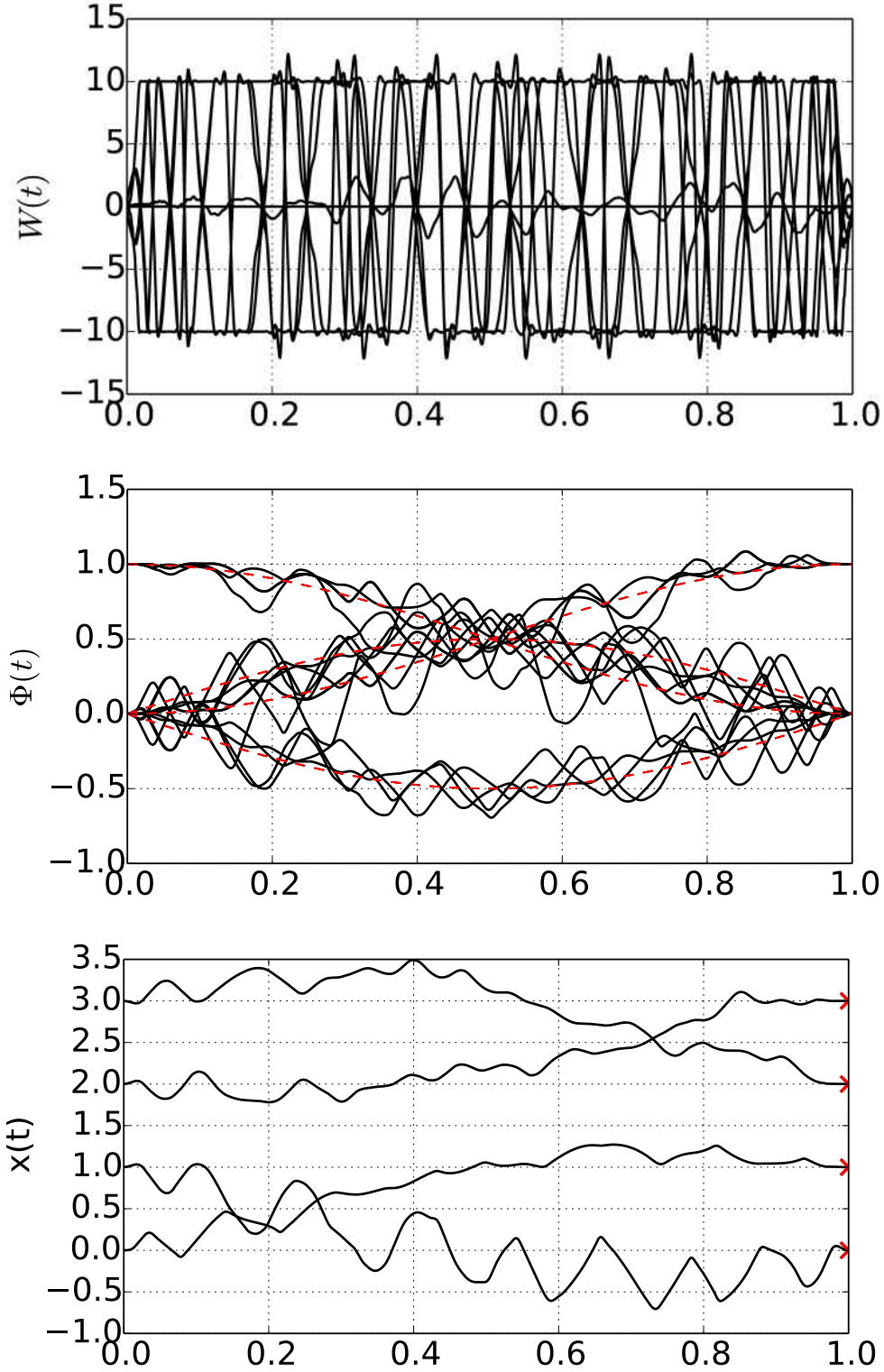


Figure 12. The time-varying weights of $W(t)$ and the trajectory of the entries of $\mathbf{X}(t)$ corresponding to a 4 node system in which the desired linear transformation swaps the values of certain nodes. In the last subfigure we show a sample evolution of node values from $x(0) = [0, 1, 2, 4]$ to $x(1) = [1, 0, 3, 2]$. The $\times$ markers indicate the entries of the desired transformation matrix T (middle) and the desired swap of state values (bottom).

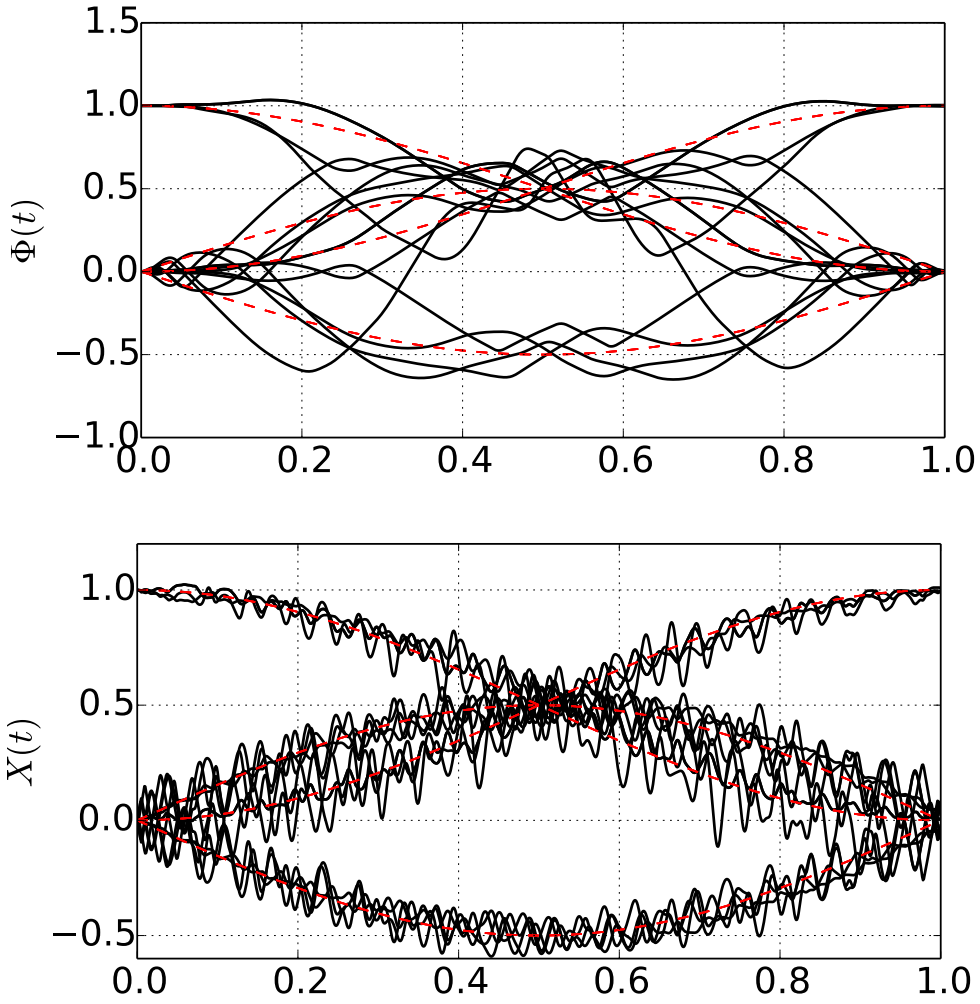


Figure 13. The tightness of the tracking of the entries of $\mathbf{X}(t)$ can be adjusted by imposing different amplitude bounds on the entries of $W(t)$; $|W_{ij}| \leq A, i, j = 1, \ldots, n$ for $A = 5$ (top) and $A = 20$ (bottom).

4.3.3 Scalability of the Pseudospectral Method

To demonstrate the size of a system this method can currently handle, several simulations were performed to determine weights that compute a random target matrix for a connected random topology. This was performed systems with a number of nodes $n = [3, 10]$. systems of less than 6 nodes complete in under a minute. above 6 nodes the time it takes to generate weighting functions increases dramatically. This trend is illustrated in Figure 14.

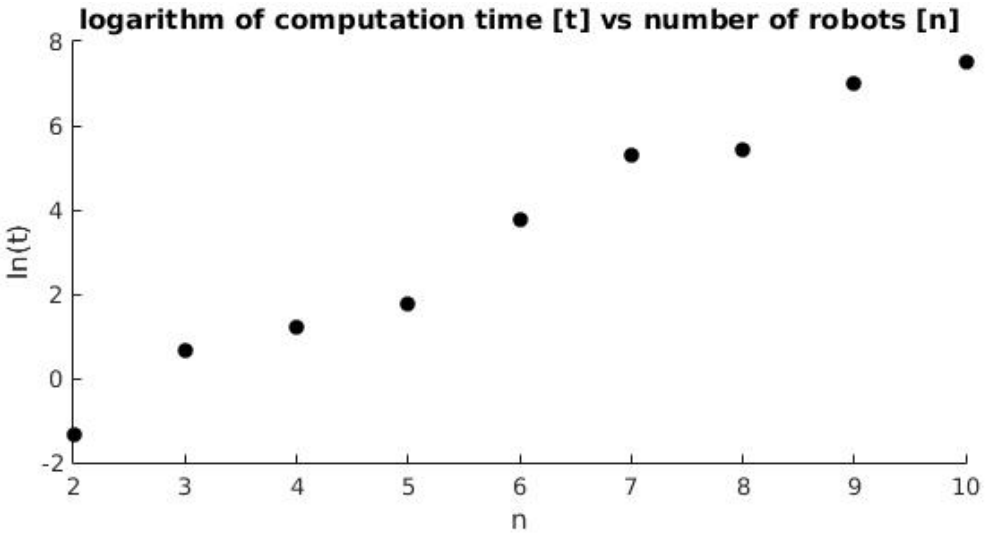


Figure 14. This plot summarizes the performance of the pseudo-spectral method as the number of robots in the system increases. The number of robots in the simulation is n. The time it takes to compute the weighted interaction rules to compute a random swap matrix in seconds is shown on the plot as t_c.

CHAPTER 5

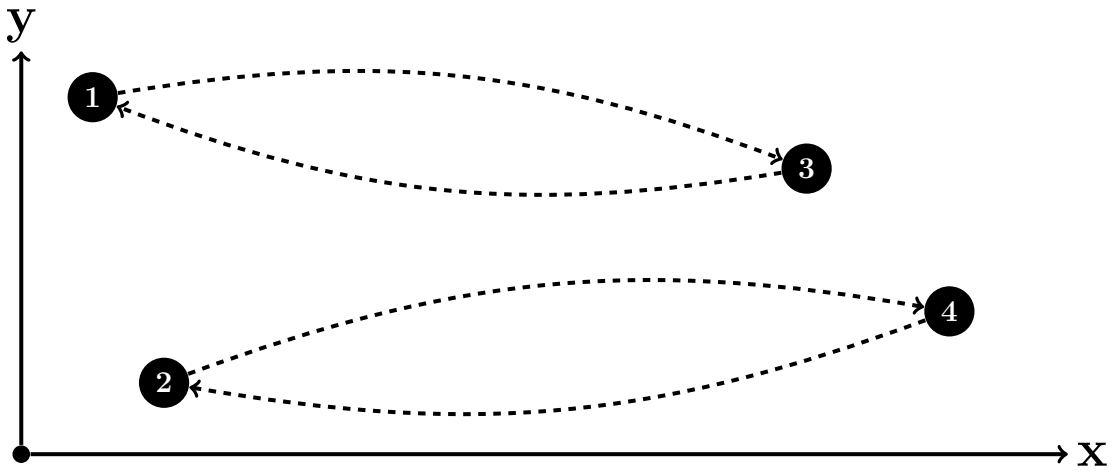
ROBOTIC IMPLEMENTATION

In networked robotic systems, information between agents can be communicated through various modalities to enable collective behavior or information processing. Two salient examples are radio communication and direct observation of neighbors' states. For example, using radio networks, strategies have been developed which attempt to recreate internet-like packet networks in distributed systems which can allow information to be shared between amongst any agents [55]. However, in the absence of availability of communication, visible information can still be collected locally by observation of nearby agents' states. This information channel, governed by the dynamics imposed on the networked robotic system can be used as a platform to perform computations on state information. These computations can enable new behaviors for networks of robots where global state information is needed, but only local communication is available.

Information exchange via physical motion is present in the natural world. For instance, bees communicate information about the direction and distance to food using a waggle dance [56]. To this end, [57] developed a technique for modulating the distances between ground robots as an information-passing scheme and were able to successfully use inter-agent distances to pass messages between connected robots.

We implement weighted interaction rules on a network of interacting robots to compute linear transformations of the robots' states. This technique can be used in any communication channel where state information can be conveyed between agents. To illustrate this claim consider an example of four unicycle robots connected in a line graph topology attempting to swap positions. A swapping transform is chosen as an example to illustrate the power of this method because it requires information to be exchanged between robots that cannot locally communicate with one another. This transformation is shown to be possible using local weighted interaction rules.

(a) An example of a 4 node interaction exchange graph. The arrows pointing into a particular node indicate what information is able to be received by that particular agent.



(b) An illustration of robots in some initial positions. The goal is to have them swap positions as indicated by the dashed lines.

Figure 15. A network of four communication constrained robots whose goal is to swap positions as illustrated by the dotted lines in (b). By executing a weight-based interaction rules we show that regardless of initial states, this information exchange will be possible.

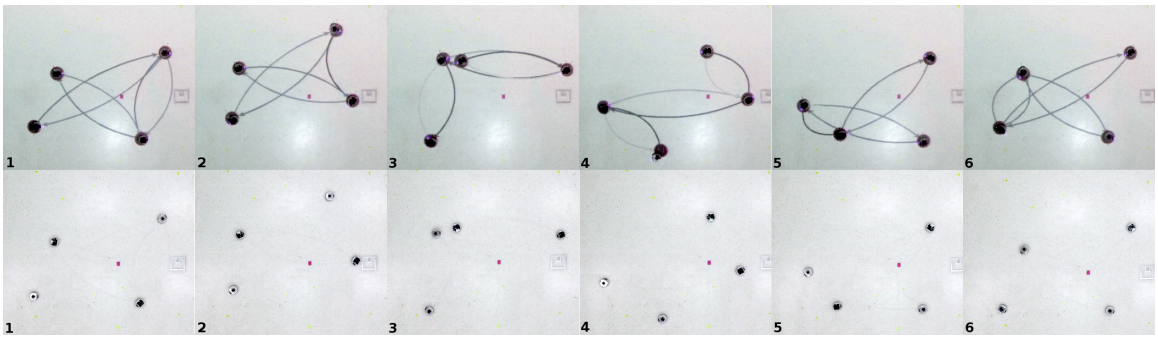


Figure 16. A sequence of images illustrating the hardware implementation of a 4 node swapping computation with Khepera III unicycle robots. Each node in the image sequence is a Khepera III unicycle robot. The lines connecting the robots are projected onto the lab floor and correspond to the magnitude of the weight on a particular edge. A darker line indicates a higher magnitude weight. The bottom images are the same transformation without the floor projection to show the Khepera III hardware. The sequence begins at the configuration on the left and proceeds right. In this figure the robot in the upper left corner in the first image swaps with the robot in the upper right corner. Additionally, the robot in the bottom left corner of the first image swaps with the robot in the bottom right corner. The weighted interaction rules can be used for any starting robot positions and will still result in the chosen swap. The swap is particularly interesting because local interactions do not provide the robots with the information required to make the swap without some coordination. For example the swap between the robot in the upper left and the lower right cannot take place directly since those two robots are not directly connected with an edge in the information exchange graph.

In Figure 15, the goal for this system is to have Robots 1 and 3 switch positions, and to have Robots 2 and 4 switch positions. Since both pairs of robots cannot directly communicate with the partner which they will swap with, it is not clear how the robots should move to accomplish this task. We use a set of weight based interaction rules associated with each communication channel as developed in the previous chapters. Weights are a natural choice for networked robotic systems because they are easily able to be assigned to adjacent robots in an any choice of connected information exchange graph. These precomputed rules, when executed result in the swapping behavior of interest under the particular choice of information exchange graph. It's important to note that these rules can be found for any connected network for many choices of linear transformation. Each set of interaction rules corresponds to a collection of robots connected by a particular information exchange graph computing one desired linear transformation.

In order to show that the distributed computation algorithm can be implemented and accurately performed, a swapping transformation was implemented on a network of 4 Khepera III unicycle robots. The target transformation that the set of agents computes is given in (87). The hardware interpretation of this transformation matrix is a position swap between agents 1 and 2, and the agents 3 and 4. The underlying information exchange graph on which this computation takes place is given by the sparsity structure of $W(t)$ in (86).

The required weights for the swapping behavior to occur were calculated in section 4.3.1.1. In order to compute a swap in $\mathbb{R}^2$, the weights must act on both the x and y coordinates of a given robots position. So, let the aggregate x and y goal coordinates of all robots be given by $x_g, y_g \in \mathbf{R}$ respectively. Let $\phi(t)$ be the mapping from the interval $[t_0, t_f]$ onto the interval $[0, 1]$. As such, the autonomous dynamics on the interval $t \in [0, 60]$ can be expressed as

$$\begin{bmatrix} \dot{x}_g(t) \\ \dot{y}_g(t) \end{bmatrix} = \begin{bmatrix} W(\phi(t)) & 0 \\ 0 & W(\phi(t)) \end{bmatrix} \begin{bmatrix} x_g(t) \\ y_g(t) \end{bmatrix} \tag{88}$$

$$\begin{bmatrix} x_g(0) \\ y_g(0) \end{bmatrix} = \begin{bmatrix} x_{g_0} \\ y_{g_0} \end{bmatrix} \tag{89}$$

where x_{g_0} and y_{g_0} are the initial x and y coordinates of the robots. Because $W(\phi(t)) \in$ sparse($\mathcal{G}$), agents can use the weights of all of the incoming edges along with the x and y coordinates of those neighbors to collaboratively compute the solution to the system given by (88). The computed $x_g(t)$ and $y_g(t)$ at a particular time are used as a goal position for a go to goal controller on the unicycle dynamics. We omit the specific form of the controller and note that any sufficiently well performing go to goal controller for a unicycle robot will work. Letting $C_v(x_g, y_g)$ and $C_\omega(x_g, y_g)$ be the controller mapping the state and goal position to the unicycle linear and angular velocity respectively, the autonomous system dynamics for a particular unicycle robot i is given by

$$\begin{aligned} \dot{x}_i &= C_v(\hat{x}_i, \hat{y}_i)\cos(\theta_i) \\ \dot{y}_i &= C_v(\hat{x}_i, \hat{y}_i)\sin(\theta_i) \\ \dot{\theta}_i &= C_\omega(\hat{x}_i, \hat{y}_i), \end{aligned} \tag{90}$$

where x_i, y_i are the physical coordinates of robot i, and θ_i is its heading. The application of these autonomous dynamics on a network of Khepera 3 unicycle robots is illustrated in the image sequence of figure 16 and the appropriate swapping behavior is observed.

Performing this swap is ultimately shows that weighted interaction rules can be implemented physically. All of the efforts in this thesis culminate in robots performing linear computations. We began by formalizing a problem whereby weighted local interaction rules would result in linear computations on node states. Existence conditions were developed. The difficulty of solving Problem 1 was discussed. Weighted interaction rules were then synthesized using various numerical methods. Then these weighted interaction rules were implemented on robotic hardware. It is our intention that this thesis provides a useful tool to synthesize global computations from local interaction rules.

CHAPTER 6

SUMMARY AND FUTURE DIRECTIONS

In this thesis we have formalized the problem of using a dynamic network with time varying weights to compute linear transforms in a distributed way. Conditions for the existence of solutions were established. A numerical approach was developed and then the method was verified in simulation. The main theorem presented states: the distributed networked system can compute a particular linear transformation T if and only if it has positive determinant. After establishing existence conditions to Problem 1, the difficulty of finding solutions was explored. Using lessons gathered from that investigation, numerical methods for computing solutions were developed and evaluated. The thesis concluded with a robotic implementation illustrating that the weighted interaction rules can be implemented on physical systems. We envision this body of work as a useful tool which can be used to compute functions of network states in a distributed way.

The original intent of this work was to answer the question: *Can a sparse dynamical network be made to behave as if it were dense?* The answer to this question is yes. We showed that we can use a sparse weight based dynamical system to compute linear transformations of the global network state. In dense networks every agent has access to the full state information of all other agents in the network. So, dense systems can easily compute functions of the full network state using information gathered from its neighbors. This investigation lead to a more fundamental question which ultimately drove the development of this thesis: *What fundamentally can be computed in a distributed way?*

This question was addressed for a nonlinear dynamical system which used weight based interaction rules and laid the groundwork for future inquiries into this question. The result was a characterization of what could be computed using weights for static graphs. This leaves the door open for several important expansions. These include investigation of computations under time varying interaction exchange graph topologies, development of

nonlinear models which are capable of universal computation, and exploration of practical distributed computing systems in unconventional mediums. We briefly comment below on each of these extensions.

The topology of the underlying information exchange graph in a network can change in some applications. One example is in power networks where tie lines may fail or new ones may be added. In these cases the precomputed weighted interaction rules will no longer have the desired effect. Instead, if interaction rules can be designed to be robust under changing graph topologies, this would greatly enhance the utility of this weight based distributed computation scheme.

One of the ultimate theoretical curiosities that motivated this work was: Can a dynamical networked system of agents be made to do universal computation? The main thrust of this work explores computation of linear transformations. For universal computation, however, nonlinear computations are needed as well. While computing linear transformations of network states have applications, it would be advantageous to compute nonlinear functions of network states in some manner. Does our existing model admit some modifications which would allow the computation of nonlinear transformations of network states?

Some systems cannot easily be outfitted with computing devices, but must in some sense perform computation in a distributed way in order to perform their function. For example, one area where this is an issue is in the field of synthetic biology. In order to engineer biological systems to be able to perform desired functions, there must be some way to translate the mathematics of a solution into a biological realization. It is still not clear how mathematical constructs such as matrix multiplication might be implemented inside of a cell [58]. Extension of the ideas presented in this thesis to support modification of biological system dynamics for computation could allow for a formal framework which would allow such a translation from mathematics to biology.

REFERENCES

[1] K. Romer and F. Mattern, "The design space of wireless sensor networks," *Wireless Communications, IEEE*, vol. 11, no. 6, pp. 54–61, 2004.

[2] F. Zhang and N. Leonard, "Coordinated patterns of unit speed particles on a closed curve," *Systems and Control Letters*, vol. 56, no. 6, pp. 397–407, 2007.

[3] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 6, pp. 926–939, 1998.

[4] M. Ji and M. Egerstedt, "Distributed coordination control of multi-agent systems while preserving connectedness," *IEEE Transactions on Robotics*, vol. 23, no. 4, pp. 693–703, 2007.

[5] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, 2003.

[6] H. Tanner, A. Jadbabaie, and G. Pappas, "Stable flocking of mobile agents, part II : Dynamic topology," in *Proc. 42nd IEEE Conf. Decision Control*, 2003.

[7] N. Michael and V. Kumar, "Controlling shapes of ensembles of robots of finite size with nonholonomic constraints," in *RSS*, 2008.

[8] M. Mesbahi and M. Egerstedt, *Graph theoretic methods in multiagent networks*. Princeton University Press, 2010.

[9] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[10] F. Bullo, J. Cortes, and S. Martnez, *Distributed Control of Robotic Networks. A Mathematical Approach to Motion Coordination Algorithms*. Princeton University Press, 2009.

[11] W. Ren and R. W. Beard, *Distributed Consensus in Multi-vehicle Cooperative Control*. Springer-Verlag, 2008.

[12] A. L. Dimeas and N. D. Hatziargyriou, "Operation of a multiagent system for microgrid control," *Power Systems, IEEE Transactions on*, vol. 20, no. 3, pp. 1447–1455, 2005.

[13] T. Ramachandran, Z. Costello, P. Kingston, S. Grijalva, and M. Egerstedt, "Distributed power allocation in prosumer networks," in *IFAC Necsys*, 2012.

[14] S. Grijalva, M. Costley, and N. Ainsworth, "Prosumer-based control architecture for the future electricity grid," in *IEEE Multi-Conference on Systems and Control*, 2011.

[15] D. Marculescu, R. Marculescu, N. H. Zamora, P. Stanley-Marbell, P. K. Khosla, S. Park, S. Jayaraman, S. Jung, C. Lauterbach, W. Weber, *et al.*, "Electronic textiles: A platform for pervasive computing," *Proceedings of the IEEE*, vol. 91, no. 12, pp. 1995–2018, 2003.

[16] J. Cortés and F. Bullo, "Coordination and geometric optimization via distributed dynamical systems," *SIAM Journal on Control and Optimization*, vol. 44, no. 5, pp. 1543–1574, 2005.

[17] A. Nedic, A. Ozdaglar, and A. Parrilo, "Constrained consensus and optimization in multi-agent networks," *IEEE Transactions on Automatic Control*, vol. 55, no. 4, pp. 922–938, 2010.

[18] R. Olfati-Saber and J. S. Shamma, "Consensus filters for sensor networks and distributed sensor fusion," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pp. 6698–6703, IEEE, 2005.

[19] T. Freeth, Y. Bitsakis, X. Moussas, J. Seiradakis, A. Tselikas, H. Mangou, M. Zafeiropoulou, R. Hadland, D. Bate, A. Ramsey, *et al.*, "Decoding the ancient greek astronomical calculator known as the antikythera mechanism," *Nature*, vol. 444, no. 7119, pp. 587–591, 2006.

[20] C. E. Shannon, "Mathematical theory of the differential analyzer," *J. Math. Phys. MIT*, vol. 20, pp. 337–354, 1941.

[21] O. Bournez and M. L. Campagnolo, "A survey on continuous time computations," in *New Computational Paradigms*, pp. 383–423, Springer, 2008.

[22] M. S. Branicky, "Universal computation and other capabilities of hybrid and continuous dynamical systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 67–100, 1995.

[23] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the national academy of sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[24] M. L. Hogarth, "Does general relativity allow an observer to view an eternity in a finite time?," *Foundations of physics letters*, vol. 5, no. 2, pp. 173–181, 1992.

[25] E. W. Blakey, *A model-independent theory of computational complexity: from patience to precision and beyond*. PhD thesis, University of Oxford, 2010.

[26] R. Sarpeshkar, "Analog versus digital: extrapolating from electronics to neurobiology," *Neural computation*, vol. 10, no. 7, pp. 1601–1638, 1998.

[27] E. Fredkin and T. Toffoli, *Conservative logic*. Springer, 2002.

[28] A. Adamatzky and B. D. L. Costello, "Experimental logical gates in a reaction-diffusion medium: The xor gate and beyond," *Physical Review E*, vol. 66, no. 4, p. 046112, 2002.

[29] L. M. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, no. 5187, pp. 1021–1024, 1994.

[30] A. Bérut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz, "Experimental verification of landauer/'s principle linking information and thermodynamics," *Nature*, vol. 483, no. 7388, pp. 187–189, 2012.

[31] R. P. Feynman, "Simulating physics with computers," *International journal of theoretical physics*, vol. 21, no. 6, pp. 467–488, 1982.

[32] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, "An autonomous molecular computer for logical control of gene expression," *Nature*, vol. 429, no. 6990, pp. 423–429, 2004.

[33] S. Sundaram and C. N. Hadjicostis, "Distributed function calculation and consensus using linear iterative strategies," *Selected Areas in Communications, IEEE Journal on*, vol. 26, no. 4, pp. 650–660, 2008.

[34] M. Rotkowitz and S. Lall, "A characterization of convex problems in decentralized control," *Automatic Control, IEEE Transactions on*, vol. 51, no. 2, pp. 274–286, 2006.

[35] J. Swigart and S. Lall, "A graph-theoretic approach to distributed control over networks," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pp. 5409–5414, IEEE, 2009.

[36] J. M. Hendrickx, R. M. Jungers, A. Olshevsky, and G. Vankeerberghen, "Graph diameter, eigenvalues, and minimum-time consensus," *Automatica*, 2013.

[37] Z. Costello and M. Egerstedt, "From global, finite-time, linear computations to local, edge-based interaction rules,"

[38] R. Brockett, *Finite Dimensional Linear Systems*. John Wiley & Sons, Inc., 1970.

[39] S. Sastry, *Nonlinear systems: analysis, stability, and control*, vol. 10. Springer New York, 1999.

[40] G. Strang, *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1993.

[41] R. W. Brockett, "System theory on group manifolds and coset spaces," *SIAM Journal on Control*, vol. 10, no. 2, pp. 265–284, 1972.

[42] R. W. Brockett *et al.*, *Asymptotic stability and feedback stabilization*. Defense Technical Information Center, 1983.

[43] F. Rüppel, G. Dirr, and U. Helmke, "Controllability of bilinear interconnected systems," *Mathematical Theory of Networks and Systems*, 2014.

[44] J. Hilgert and K.-H. Neeb, *Structure and geometry of Lie groups*. Springer Science & Business Media, 2011.

[45] F. Hiai and D. Petz, "Riemannian metrics on positive definite matrices related to means," *Linear Algebra and its Applications*, vol. 430, no. 11, pp. 3105–3130, 2009.

[46] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[47] D. Liberzon, *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press, 2012.

[48] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

[49] A. Bhaya, "Real matrices with positive determinant are homotopic to the identity," *SIAM review*, vol. 40, no. 2, pp. 335–340, 1998.

[50] F. Fahroo and I. M. Ross, "Pseudospectral methods for infinite-horizon nonlinear optimal control problems," *Journal of Guidance Control and Dynamics*, vol. 31, no. 4, pp. 927–936, 2008.

[51] J. Ruths and J.-S. Li, "Optimal Control of Inhomogeneous Ensembles," *Automatic Control, IEEE Transactions on*, vol. 57, no. 8, pp. 2021–2032, 2012.

[52] J. Ruths, P. N. Taylor, and J. Dauwels, "Optimal Control of an Epileptic Neural Population Model," in *International Federation of Automatic Control, Cape Town*, pp. 3116–3121, IFAC, 2014.

[53] R. Fourer, D. M. Gay, and B. W. Kernighan, "AMPL: A mathematical programming language," 2002.

[54] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Knitro: An Integrated Package for Nonlinear Optimization - Springer," *Large-scale nonlinear optimization*, 2006.

[55] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, *Mobile Ad Hoc networking: the cutting edge directions*, vol. 35. John Wiley & Sons, 2013.

[56] A. Michelsen, "The transfer of information in the dance language of honeybees: progress and problems," *Journal of Comparative Physiology A*, vol. 173, no. 2, pp. 135–141, 1993.

[57] D. Raghunathan and J. Baillieul, "Motion based communication channels between mobile robots-a novel paradigm for low bandwidth information exchange," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 702–708, IEEE, 2009.

[58] E. Klavins, "Proportional-integral control of stochastic gene regulatory networks," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 2547–2553, IEEE, 2010.