Clemson University
TigerPrints

All Theses Theses

5-2017

Phase Desynchronization in Pulse-Coupled Oscillator Networks: A New Algorithm and Approach

Timothy Benjamin Anglea
Clemson University, mathman93@gmail.com

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Recommended Citation
Anglea, Timothy Benjamin, "Phase Desynchronization in Pulse-Coupled Oscillator Networks: A New Algorithm and Approach" (2017). *All Theses*. 2597.
https://tigerprints.clemson.edu/all_theses/2597

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

Phase Desynchronization in Pulse-Coupled Oscillator Networks: A New Algorithm and Approach

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Timothy Benjamin Anglea
May 2017

Accepted by:
Dr. Yongqiang Wang, Committee Chair
Dr. Richard Groff
Dr. Ian D. Walker

Abstract

In this thesis, we present a novel approach for achieving phase desynchronization in a pulse-coupled oscillator network. Ensuring phase desynchronization is a difficult problem, and existing results are constrained to a completely interconnected network and a fixed number of oscillators. Our approach is more robust than previous approaches, removing the constraint of a fixed number of oscillators. The removal of this constraint is significant because it allows the network to receive and drop nodes freely without any change to the phase update strategy. Also, to our knowledge, our approach is the first to prove the convergence to the desynchronized state for a topology that is more general than the all-to-all topology. More specifically, our approach is applicable to any circulant and symmetric network topology, including the circulant symmetric ring topology. Rigorous mathematical proofs are provided to support the result that any circulant symmetric network with ordered phases under our proposed algorithm will converge to uniform phase desynchronization. Simulation results are presented to demonstrate the algorithm's performance, as well as experimental results on a physical system to further illustrate applications of pulse-coupled oscillator networks.

Dedication

I would like to dedicate this thesis to my parents, Steven Mark and Patricia Ruth Anglea, who have been an encouragement to me throughout my academic pursuits in higher education.

Acknowledgments

I would first like to thank my advisor and professor at Clemson University, Dr. Yongqiang Wang, for introducing me to the concepts and applications of pulse-coupled oscillators. His help and guidance in my research endeavors has been invaluable. I must also thank the undergraduate students of the Robotic Networks Creative Inquiry that Dr. Wang leads, who have helped immensely in the development and testing of the various aspects of the Roomba platform, on which experimental results of pulse-coupled oscillators have been achieved.

Additional thanks go to Dr. Richard Groff and Dr. Ian Walker of the Electrical & Computer Engineering Department at Clemson University for their help in reviewing this thesis and their willingness to aid me in my degree as committee members.

I would also like to thank all others who reviewed the initial drafts of this thesis and the works leading up to it. Their insights and corrections are greatly appreciated.

Contents

Title Page . . . . . **i**

Abstract . . . . . **ii**

Dedication . . . . . **iii**

Acknowledgments . . . . . **iv**

List of Figures . . . . . **vi**

1 Introduction **1**

2 Phase Desynchronization **4**
2.1 Network Dynamics 4
2.2 All-to-All Case 9
2.3 Circulant Symmetric Case 15

3 Simulation Results **20**
3.1 Proposed Algorithm Results 20
3.2 Example of Insufficient Initial Condition 22
3.3 Comparison with DESYNC-STALE 23
3.4 Proposed Algorithm Convergence 25

4 Experimental Results **30**
4.1 Roomba Robot Network 30
4.2 All-to-all Topology Results 33
4.3 General Topology Results 35
4.4 Discussion 36

5 Conclusion **39**

Appendices **41**
A Proofs of Lemmas 42
B Arduino Desynchronization Code for Roombas 45

Bibliography **61**

List of Figures

2.1 General state of pulse-coupled oscillator phases 5
2.2 Phase Responce Curve (PRC) for the proposed algorithm 6
2.3 Examples of circulant symmetric networks 8
2.4 Illustration of equation (2.13) 12

3.1 Phase evolution of the proposed algorithm in all-to-all and circulant symmetric topologies 21
3.2 Phase differences of the proposed alogirthm in all-to-all and circulant symmetric topologies 22
3.3 Change value evolution of the proposed algorithm in all-to-all and circulant symmetric topologies 23
3.4 Network energy as a function of time for the proposed algorithm in all-to-all and circulant symmetric topologies 24
3.5 Network energy after a complete firing in all-to-all and circulant symmetric topologies 25
3.6 Phase evolution under the proposed algorithm with insufficient initial conditions . . 26
3.7 Phase differences under the proposed algorithm with insufficient initial conditions . . 26
3.8 Network energy as a function of time under the proposed algorithm with insufficient initial conditions 27
3.9 Phase differences of the proposed algorithm and the DESYNC-STALE algorithm . . 27
3.10 Desynchronization accuracy of the proposed algorithm and the DESYNC-STALE algorithm 28
3.11 Convergence rate of the network under the proposed algorithm 29

4.1 A single Roomba robot 31
4.2 External Roomba sensors 32
4.3 Comparison of Roomba phase and heading between simulation and experimental data in an all-to-all network topology 33
4.4 Comparison of Roomba phase differences between simulation and experimental data in an all-to-all network topology 34
4.5 Comparison of Roomba change values and network energy between simulation and experimental data in an all-to-all network topology 35
4.6 Comparison of Roomba phase and heading between simulation and experimental data in a star network topology 36
4.7 Comparison of Roomba phase differences between simulation and experimental data in a star network topology 37
4.8 Comparison of Roomba change values and network energy between simulation and experimental data in a star network topology 38

Chapter 1

Introduction

The study and analysis of pulse-coupled oscillators (PCOs) is a currently active field of engineering research. The pulse-coupled oscillator (PCO) model was first introduced by Peskin in 1975 [1].

The basic concept of a pulse-coupled oscillator network is simple. Each node in the network has a continuously cycling phase variable, or oscillator, that increases at a constant rate. When this oscillator reaches some threshold value, it resets its phase to zero and it sends, or fires, a single, simple pulse signal to any connected nodes in the network. When a node receives a pulse signal, it then modifies is phase variable according to some control algorithm. Each oscillator sends an identical pulse, often just a single bit, thus allowing the control algorithm to be the same for each node in the network.

Peskin used PCOs to model the synchronization of pacemaker cells in the heart [1]. PCOs are also used to model the synchronization of many other biological systems, such as the firing of neurons [1, 2], and the flashing of fireflies [3, 4]. In synchronization, the oscillators in the network attempt to converge to an identical phase value through the resulting pulse-coupled interaction between the nodes.

A few years after Peskin's introduction of the PCO model, Mirollo and Strogatz improved the model, providing a more rigorous mathematical formulation [3]. Up to the present time, the synchronizing PCO model has been used and developed by many others [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], who have presented ways to improve the model's capabilities and characteristics, and extend its applications. The primary benefits of the model are found through the minimization of

communication latency, packet loss, signal corruption, and energy consumption, and are due to the simplicity of the communication between nodes in the network, i.e., single pulses.

More recently, the model has been expanded to encompass more complex behaviors other than synchronization [17, 18]. One such behavior is desynchronization, a more difficult problem and the inverse problem to synchronization. In desynchronization, the phases of the oscillators attempt to spread apart evenly from each other, such that there is equal separation between all oscillators. This state of desynchronization is sometimes termed the splay state.

Desynchronization has many important applications, primarily in the area of decentralized network communication [19, 20, 21], but also in the design of analog-to-digital converters and the control of traffic intersection flow [19]. Compared with the increased research on achieving oscillator synchronization, as discussed above, there has been much less research on desynchronization in PCO networks.

Many algorithms and approaches have been proposed to achieve desynchronization. Some control strategies ensure weak desynchronization, where the timing between oscillator firings is constant, but the phases are constantly shifting [19, 22]. Other control strategies ensure strong, or phase, desynchronization, but must assume a completely interconnected, or all-to-all, network and a fixed number of nodes [19, 20, 21, 23, 24]. Some have tried to relax the constraint of an all-to-all connection topology, but instead require either a continuously coupled, rather than pulse-coupled, network [17, 25], or the use of additional types of nodes and pulse transmissions [14]. Work has also been done to improve the speed of convergence assuming an all-to-all topological network [26, 27].

In this thesis, we propose a desynchronization strategy and algorithm that relaxes the constraint of complete interconnection and removes the requirement of a fixed number of nodes to achieve complete phase desynchronization. To the best of our knowledge, our approach is the first to prove that a network topology other than the all-to-all topology can be used to achieve uniform phase desynchronization with identical pulse-coupled oscillators, and is unique to the literature for desynchronizing pulse-coupled oscillator networks.

Inspiration for the proposed algorithm is based on the DESYNC and DESYNC-STALE algorithms proposed in [19]. However, different from [19], which only allows a node to react based on the oscillators directly ahead of and behind it in phase, our approach allows every node to react to all nodes with which it is connected.

A mechanical analogy can be made, where each node, spread around the unit circle, is

connected to other nodes by compressed springs [9]. The closer the nodes are in phase, the more force the nodes will exert in moving the other away. An equilibrium is achieved when all of the nodes are as far away from each other in phase as possible, the desired state of desynchronization.

In Chapter 2, we will formally introduce the proposed phase desynchronization algorithm. We provide rigorous proof that the strategy guarantees phase desynchronization in a completely connected network, as well as the class of circulant symmetric networks. In Chapter 3, we will simulate our proposed algorithm, showing how the network desynchronizes, and analyzing the algorithm's convergence rate. We will then apply the proposed algorithm to a real system in Chapter 4, and compare the results to the simulations. We will offer final conclusions in Chapter 5.

Chapter 2

Phase Desynchronization

2.1 Network Dynamics

2.1.1 General Algorithm Definitions

Let us assume a network of N pulse-coupled oscillators, as shown in Fig. 2.1. Let $\theta_i \in [0, 2\pi]$ be the associated phase of oscillator, or node, $i \in \mathcal{V} = \{1, 2, \cdots, N\}$, with ω as the fundamental frequency for all oscillators. The state of the network $\vec{\theta} = [\theta_i]$ is a column vector representing the phase of the N nodes.

We define $\mathcal{A}_{\mathcal{R}} = [a_{ij}] \in \mathbb{R}^{N \times N}$ to be the adjacency matrix for the network, where element a_{ij} is 1 if there is a connection from node i to node j, and 0 if there is no connection. The adjacency matrix defines the topology for the network. $\mathcal{A}_{\mathcal{R}}$ is zero diagonal ($a_{ii} = 0 \; \forall i \in \mathcal{V}$), and is symmetrical if connections between nodes are bidirectional. For each node i, we let $\mathcal{N}_i$ denote the set of nodes connected to node i, where $\mathcal{N}_i \subset \mathcal{V}$, and $|\mathcal{N}_i| = n_i$.

Our proposed algorithm[1]can be described as follows.

1. A node fires a pulse when its phase reaches 2π. When the node fires, all connected nodes receive the pulse.

2. When a node receives a pulse, the node records the value of *its own* current phase, $\theta_{i,m}$. This value is stored to be used when the node fires, i.e., when its phase reaches 2π.

[1]This algorithm has been published in the IEEE Transactions on Signal Processing, Vol. 65, No. 5. [28].

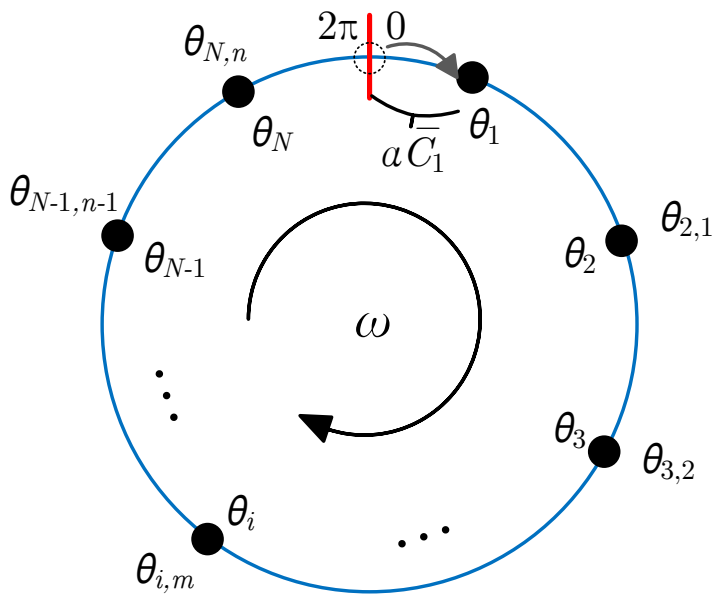


Figure 2.1: A snapshot of current and recorded phases when a node fires for a network of N pulse-coupled oscillators with fundamental frequency ω. When a node fires, all connected nodes record the mth instance of their current phase. For example, when node 1 fires, node 2 records $\theta_{2,1}$ and node 3 records $\theta_{3,2}$. Node 3 recorded $\theta_{3,1}$ when node 2 fired previously.

3. When a node fires, emits a pulse, and resets its phase to zero, it jumps, and the amount that the node jumps is dependent upon all of the recorded phase values due to the firing of the other nodes. The node computes the phase response curve (PRC) value, $Q(\theta_{i,m})$, for each of its own recorded phase values, and then computes the change value, C_i, or the amount that the node needs to jump, from the PRC values. If the change value is positive, the node will jump by a proportional amount, αC_i, where α is a positive real number; otherwise, the node will not jump. Thus, the node will only jump forward in phase.

The phase update for the network is asynchronous. Each node updates its phase when it fires, and does not wait for other nodes. However, the amount that each node jumps is determined by the recorded phase values found from the previous firing cycle. In other words, the network relies on "stale" information obtained from the timing of received pulses, as in the DESYNC-STALE algorithm in [19]. In this paper, the invariance of the firing sequence of the nodes is assumed, which will be proven later.

Let us denote the stroboscopic recorded phase values when a node receives a pulse as $\theta_{i,m}$, where i is the node that received the pulse, and $m \in \{1, 2, \ldots, n_i\}$ is the index of the recorded phase

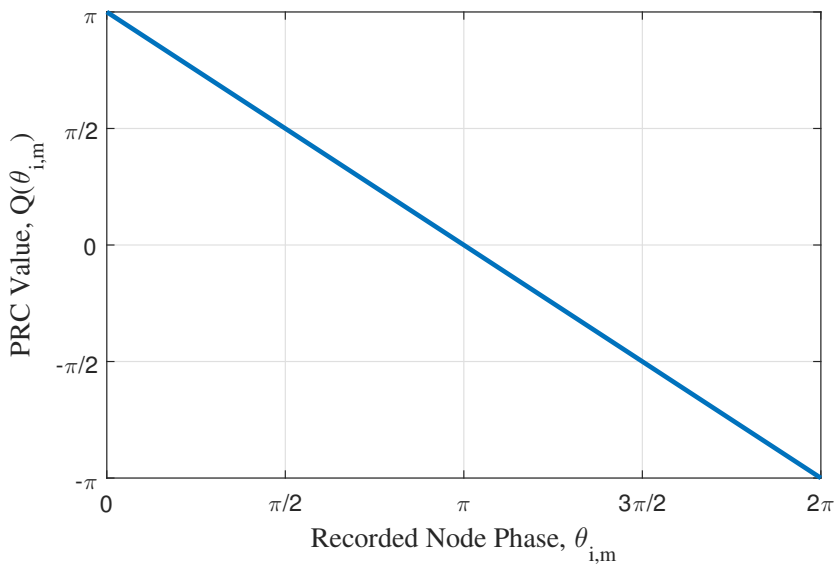


Figure 2.2: Phase Response Curve, Q, as given in (2.1) for the proposed algorithm.

value of that node due to the pulses received from neighboring nodes in $\mathcal{N}_i$. Thus, $0 < \theta_{i,1} < \theta_{i,2} < \ldots < \theta_{i,n_i} < 2\pi$.

We define the PRC of the network, Q, in the following way:

$$Q(\theta_{i,m}) = -(\theta_{i,m}) + \pi \tag{2.1}$$

An interpretation of the PRC function, as shown in Fig. 2.2, is that if two nodes are close in phase, they will exert a force on each other that causes them to move farther away from each other. Otherwise, when the nodes are as far apart as possible ($\theta_{i,m} = \pi$), they will exert no force on each other.

2.1.2 Pulse Dynamics

The amount that a node jumps is dependent on the relative position and total number of other nodes from which it has received pulses. For our algorithm, the node will compute the average amount of the calculated PRC values from its recorded phase values. Let us, then, define the change

value, C_i, to be this average amount:

$$C_i = \frac{1}{n_i} \sum_{m=1}^{n_i} Q(\theta_{i,m}) \tag{2.2}$$

As mentioned earlier, when a node fires, it computes its change value and then jumps by a portion of that amount if the change value is positive. Otherwise, the node does not jump. This jump rule is described mathematically as

$$\theta_i^+ = \begin{cases} \alpha C_i & \text{if } C_i > 0 \\ 0 & \text{if } C_i \leq 0 \end{cases} \tag{2.3}$$

To summarize, a node will jump in phase only when it fires, and then only if it will jump forward in phase.

Remark 1. *In application, the phase could be allowed to jump backward, allowing for faster convergence. However, there will exist the problem of having the state, in theory, be less than zero for a period of time. Care will need to be taken to ensure the proper behavior of the network and to maintain the proper firing sequence.*

Let us define the change vector, C, as the column vector of the phase change values corresponding to all nodes in the network:

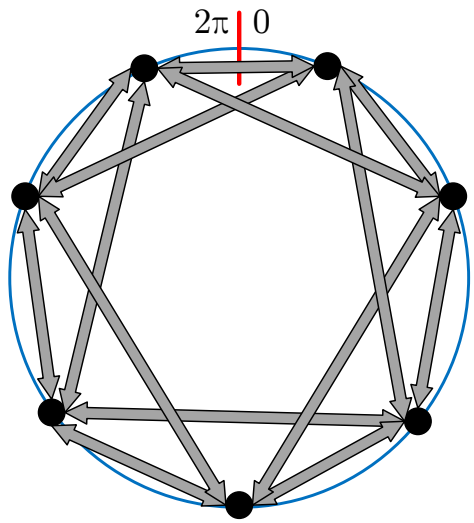
$$C = [C_i] \in \mathbb{R}^N \tag{2.4}$$

2.1.3 Network Equilibrium

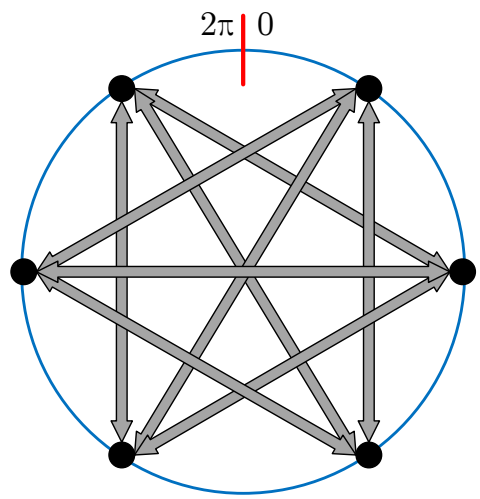
In order for us to show that the network will desynchronize, we need to show that the splay state is the only equilibrium point for the network under our proposed algorithm. For this paper we will show that the splay state is the equilibrium point for any circulant symmetric network with ordered nodes.

Definition 1. *Nodes in a PCO network are considered ordered when they are initially numbered in order by their phase magnitude, i.e.,* $0 \leq \theta_1 < \theta_2 < \ldots < \theta_{N-1} < \theta_N \leq 2\pi$.

Definition 2. *A network with ordered nodes is called circulant symmetric if the resulting adjacency matrix* $\mathcal{A}_\mathcal{R}$ *for the ordered nodes is both circulant and symmetric.*



(a) Seven node network, $N = 7$, with each node connected to two nodes ahead in phase and two nodes behind in phase.



(b) Six node network, $N = 6$, where each node is without connections to the nodes directly ahead and behind it in phase.

Figure 2.3: Examples of circulant symmetric networks that will achieve desynchronization under the proposed algorithm.

Remark 2. *For our paper, a circulant symmetric network assumes ordered nodes.*

Remark 3. *For circulant symmetric networks, all nodes have the same number of neighbors, or* $\mathcal{N}_i = n \ \forall i \in \mathcal{V}$. *For the all-to-all network topology,* $n = N - 1$, *and for the ring topology,* $n = 2$.

Proposition 1. *For any circulant symmetric network under the proposed algorithm, the network is desynchronized if and only if the change vector in (2.4) is the zero vector.*

Proof. When the network is in the state of desynchronization, all nodes are as far apart from each other in phase as possible. In this state, the phases are equally spaced around the region $[0, 2\pi]$. Since the network is circulant and symmetric, for each node in the network, there will be equal force pushing the node forward in phase as there is pushing it backward in phase. Thus, over a cycle of firings, C_i will be zero for all nodes, and no node will jump in phase.

If the network is not desynchronized, then since the network is circulant and symmetric, there will be an imbalance in the network, causing the change vector to be non-zero. □

Proposition 1 indicates that the desynchronized state is the only equilibrium state under our setup. This result gives us insight into how to determine when the network has reached desynchronization.

2.1.4 Network Energy Measure

Let us define an energy function for the network using the change vector after the kth firing cycle:

$$p(k) = \frac{1}{2}C(k)^T C(k) = \frac{1}{2}\sum_{i=1}^{N} C_i(k)^2 \tag{2.5}$$

By the definition, the energy is non-negative everywhere, and zero only when the change vector from (2.4) is the zero vector. Thus, (2.5) is a potential "Lyapunov" function. The energy is computed after every firing cycle, when all nodes have fired once in the sequence.

We also need to define the change in the energy as a function of the cycle:

$$\Delta p(k) = p(k+1) - p(k) \tag{2.6}$$

Proposition 2. *For any circulant symmetric network under the proposed algorithm, the network will desynchronize if $\Delta p(k) < 0$ for any k when $p(k) > 0$.*

Using LaSalle's theorem given in [29], Proposition 2 follows easily from the defined energy function and Proposition 1.

In the next sections, we will prove that the condition in Proposition 2 holds for the all-to-all topology and circulant symmetric topologies in general, specifically for the circulant symmetric ring topology.

Remark 4. *The all-to-all and circulant symmetric ring topologies are special cases of circulant symmetric networks. Networks with adjacency matrices that are both symmetric and circulant are a subset of highly symmetrical networks, as described in [9].*

2.2 All-to-All Case

2.2.1 Preliminaries

In order for the network to desynchronize, we need to ensure that the firing sequence of the nodes remains unchanged between cycles. If nodes switch order after a cycle, then the resulting network with ordered nodes may no longer be circulant.

Lemma 1. *The firing sequence for any circulant symmetric network is invariant between cycles under the proposed algorithm for any change ratio* $\alpha \in (0, 1]$.

See Appendix A for proof.

To simplify our proofs, we will utilize the following property of the change values C_i.

Lemma 2. *For a circulant symmetric network under the proposed algorithm,* $\sum_{i=1}^{N} C_i = 0$ *holds if no nodes have jumped during the previous cycle.*

See Appendix A for proof.

Remark 5. *For both Lemma 1 and Lemma 2, it is necessary that the network be circulant symmetric. Thus, we restrict our analysis to this class of networks.*

2.2.2 Convergence of the All-to-All Topology

Let us first consider a PCO network with an all-to-all topology. This network has been shown to converge under other desynchronization algorithms [19], so we will show that the network will also converge under our proposed algorithm.

Theorem 1. *A network with an all-to-all topology under the proposed algorithm will converge to the desynchronized state.*

Proof. We can prove Theorem 1 if we can show that $\Delta p(k) < 0$ for a general k when $p(k)$ is non-zero.

Let us first note that the expression for the change in energy can be rewritten by combining (2.5) and (2.6) and simplifying:

$$\Delta p(k) = \frac{1}{2} \sum_{i=1}^{N} \Big[C_i(k+1)^2 - C_i(k)^2 \Big] \tag{2.7}$$

By substituting the PRC in (2.1) into (2.2), we can write an expression for the change value for a node at a given cycle:

$$C_i(k) = \frac{1}{n} \sum_{m=1}^{n} \big(-\theta_{i,m}(k) + \pi \big) \tag{2.8}$$

Let $j \in \mathcal{N}_i$ be an element in the set of nodes that are connected to node i. For the all-to-all topology, $\mathcal{N}_i$ includes all of the network nodes except node i. The recorded phase values can be expressed as

$$\theta_{i,m}(k) = 2\pi - \Big(\big(\theta_j(k) - \theta_i(k) \big) \mod 2\pi \Big), \tag{2.9}$$

where j corresponds to the node that resulted in the mth recorded phase value. We can thus express (2.8) in terms of the node phases during the current cycle as follows:

$$C_i(k) = \frac{1}{n} \sum_{j \in \mathcal{N}_i} \Big[\Big(\big(\theta_j(k) - \theta_i(k)\big) \mod 2\pi \Big) - \pi \Big] \tag{2.10}$$

Similarly, the change value for the same node at the next cycle can also be expressed in terms of the node phases during the next cycle:

$$\begin{aligned} C_i(k+1) &= \frac{1}{n} \sum_{m=1}^{n} \big(-\theta_{i,m}(k+1) + \pi \big) \\ &= \frac{1}{n} \sum_{j \in \mathcal{N}_i} \Big[\Big(\big(\theta_j(k+1) - \theta_i(k+1)\big) \mod 2\pi \Big) - \pi \Big] \end{aligned} \tag{2.11}$$

We now need an expression for the recorded phase value for the next cycle. We know that, in the next cycle, the phases may have jumped a fractional amount of C_i, which may have been non-positive, from the previous cycle following the update rule given in (2.3). The amount that node i jumps during the current cycle, which we will denote as $\bar{C}_i$, can be mathematically described as the maximum of zero and C_i, or

$$\bar{C}_i = \max\{0, C_i\} \tag{2.12}$$

Eqn. (2.12) allows us to write an expression for the next recorded phase value based on the previous recorded phase value and the amount that each node jumped in the previous cycle, as illustrated in Fig. 2.4:

$$\theta_{i,m}(k+1) = \theta_{i,m}(k) + \alpha\big(\bar{C}_i(k) - \bar{C}_j(k)\big) \tag{2.13}$$

Using the expression in (2.9), we can rewrite (2.13) as

$$\begin{aligned} 2\pi - \Big(\big(\theta_j(k+1) - \theta_i(k+1)\big) \mod 2\pi \Big) &= 2\pi - \Big(\big(\theta_j(k) - \theta_i(k)\big) \mod 2\pi \Big) + \alpha\big(\bar{C}_i(k) - \bar{C}_j(k)\big) \\ \implies \Big(\big(\theta_j(k+1) - \theta_i(k+1)\big) \mod 2\pi \Big) &= \Big(\big(\theta_j(k) - \theta_i(k)\big) \mod 2\pi \Big) + \alpha\big(\bar{C}_j(k) - \bar{C}_i(k)\big) \end{aligned} \tag{2.14}$$

Thus, the expression for the next change value, by substituting (2.14) into (2.11), can be

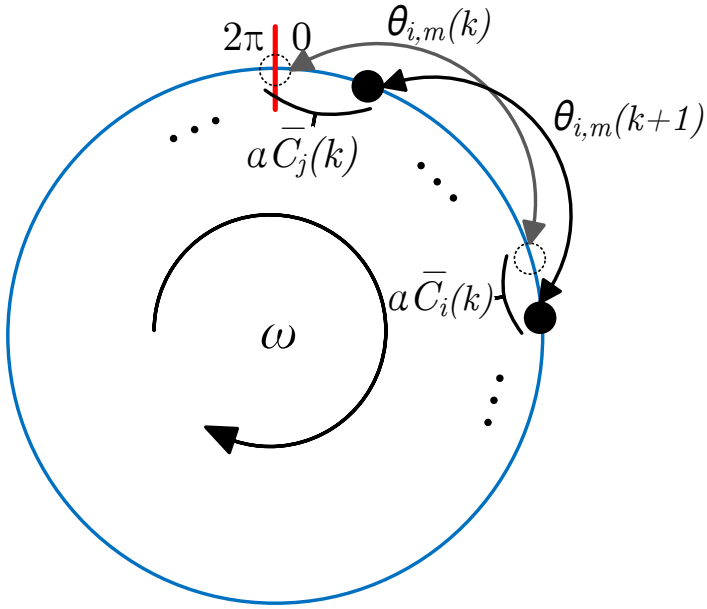


Figure 2.4: Illustration for (2.13), showing the change in $\theta_{i,m}$ from one cycle to the next.

written as

$$C_i(k+1) = \frac{1}{n}\sum_{j\in\mathcal{N}_i}\Big[\Big(\big(\theta_j(k)-\theta_i(k)\big) \mod 2\pi\Big) + \alpha\big(\bar{C}_j(k)-\bar{C}_i(k)\big) - \pi\Big]$$
$$= \frac{1}{n}\sum_{j\in\mathcal{N}_i}\Big[\Big(\big(\theta_j(k)-\theta_i(k)\big) \mod 2\pi\Big) - \pi\Big] + \frac{\alpha}{n}\sum_{j\in\mathcal{N}_i}\big[\bar{C}_j(k)-\bar{C}_i(k)\big] \quad (2.15)$$

Recognizing the expression for the change value from (2.10), we can simplify (2.15) as follows:

$$C_i(k+1) = C_i(k) + \frac{\alpha}{n}\sum_{j\in\mathcal{N}_i}\big[\bar{C}_j(k)-\bar{C}_i(k)\big] \quad (2.16)$$

Squaring both sides of (2.16) and subtracting $C_i(k)^2$ from both sides give us the following expression for the summation term in (2.7):

$$C_i(k+1)^2 - C_i(k)^2 = \frac{2\alpha}{n}C_i(k)\sum_{j\in\mathcal{N}_i}\big[\bar{C}_j(k)-\bar{C}_i(k)\big] + \Big(\frac{\alpha}{n}\sum_{j\in\mathcal{N}_i}\big[\bar{C}_j(k)-\bar{C}_i(k)\big]\Big)^2 \quad (2.17)$$

We can rearrange and simplify (2.17) as follows:

$$
\begin{aligned}
&C_i(k+1)^2 - C_i(k)^2 \\
&\qquad = \frac{2\alpha}{n}\Big[C_i(k)\sum_{j\in\mathcal{N}_i}\bar{C}_j(k) - C_i(k)\sum_{j\in\mathcal{N}_i}\bar{C}_i(k)\Big] + \frac{\alpha^2}{n^2}\Big(\sum_{j\in\mathcal{N}_i}\big[\bar{C}_j(k) - \bar{C}_i(k)\big]\Big)^2 \\
&\qquad\qquad = \frac{2\alpha}{n}\Big[C_i(k)\sum_{j\in\mathcal{N}_i}\bar{C}_j(k) - nC_i(k)\bar{C}_i(k)\Big] + \frac{\alpha^2}{n^2}\Big(\sum_{j\in\mathcal{N}_i}\big[\bar{C}_j(k)\big] - n\bar{C}_i(k)\Big)^2 \qquad (2.18)
\end{aligned}
$$

We can use the fact that $C_i(k)\bar{C}_i(k) = \bar{C}_i(k)^2$, and then expand and recombine (2.18) to achieve the following:

$$
\begin{aligned}
&C_i(k+1)^2 - C_i(k)^2 \\
&\quad = \frac{2\alpha}{n}C_i(k)\sum_{j\in\mathcal{N}_i}\bar{C}_j(k) - 2\alpha\bar{C}_i(k)^2 + \frac{\alpha^2}{n^2}\Big[\sum_{j\in\mathcal{N}_i}\bar{C}_j(k)\Big]^2 - \frac{2\alpha^2}{n}\bar{C}_i(k)\sum_{j\in\mathcal{N}_i}\bar{C}_j(k) + \alpha^2\bar{C}_i(k)^2 \\
&\qquad = (\alpha^2 - 2\alpha)\big(\bar{C}_i(k)^2\big) + \frac{\alpha^2}{n^2}\Big[\sum_{j\in\mathcal{N}_i}\bar{C}_j(k)\Big]^2 + \Big(\frac{2\alpha}{n}C_i(k) - \frac{2\alpha^2}{n}\bar{C}_i(k)\Big)\Big[\sum_{j\in\mathcal{N}_i}\bar{C}_j(k)\Big] \qquad (2.19)
\end{aligned}
$$

Substituting (2.19) into (2.7) gives us an expression for the change in network energy, where we drop the cycle (k) notation for clarity:

$$
\begin{aligned}
\Delta p &= \frac{1}{2}\sum_{i=1}^{N}\Big[(\alpha^2 - 2\alpha)(\bar{C}_i^2) + \frac{\alpha^2}{n^2}\big(\sum_{j\in\mathcal{N}_i}\bar{C}_j\big)^2 + \Big(\frac{2\alpha}{n}C_i - \frac{2\alpha^2}{n}\bar{C}_i\Big)\big(\sum_{j\in\mathcal{N}_i}\bar{C}_j\big)\Big] \\
&= \frac{1}{2}\Bigg((\alpha^2 - 2\alpha)\sum_{i=1}^{N}(\bar{C}_i^2) + \frac{\alpha^2}{n^2}\sum_{i=1}^{N}\Big[\sum_{j\in\mathcal{N}_i}\bar{C}_j\Big]^2 + \sum_{i=1}^{N}\Big[\Big(\frac{2\alpha}{n}C_i - \frac{2\alpha^2}{n}\bar{C}_i\Big)\big(\sum_{j\in\mathcal{N}_i}\bar{C}_j\big)\Big]\Bigg) \qquad (2.20)
\end{aligned}
$$

We can further simplify the second term in (2.20) by expanding and recombining terms as follows:

$$
\sum_{i=1}^{N}\big(\sum_{j\in\mathcal{N}_i}\bar{C}_j\big)^2 = n\sum_{i=1}^{N}\bar{C}_i^2 + (n-1)\Big(\sum_{i=1}^{N}\bar{C}_i\big(\sum_{j\in\mathcal{N}_i}\bar{C}_j\big)\Big) \qquad (2.21)
$$

Substituting (2.21) into (2.20) and recombining terms give the following result:

$$\begin{aligned}\Delta p = \frac{1}{2}\Bigg(&(\alpha^2 - 2\alpha)\sum_{i=1}^{N}(\bar{C}_i^2) + \frac{\alpha^2}{n^2}\Big(n\sum_{i=1}^{N}\bar{C}_i^2 + (n-1)\Big[\sum_{i=1}^{N}\bar{C}_i(\sum_{j\in\mathcal{N}_i}\bar{C}_j)\Big]\Big) \\ &+ \sum_{i=1}^{N}\Big[\Big(\frac{2\alpha}{n}C_i - \frac{2\alpha^2}{n}\bar{C}_i\Big)(\sum_{j\in\mathcal{N}_i}\bar{C}_j)\Big]\Bigg) \\ = \frac{1}{2}\Bigg(&(\alpha^2 - 2\alpha)\sum_{i=1}^{N}(\bar{C}_i^2) + \frac{\alpha^2}{n}\sum_{i=1}^{N}(\bar{C}_i^2) + \sum_{i=1}^{N}\Big[\frac{\alpha^2(n-1)}{n^2}\bar{C}_i(\sum_{j\in\mathcal{N}_i}\bar{C}_j)\Big] \\ &+ \sum_{i=1}^{N}\Big[\Big(\frac{2\alpha}{n}C_i - \frac{2\alpha^2}{n}\bar{C}_i\Big)(\sum_{j\in\mathcal{N}_i}\bar{C}_j)\Big]\Bigg) \qquad (2.22)\end{aligned}$$

Further recombination and simplification lead to the following final result:

$$\Delta p = \sum_{i=1}^{N}\Big[\Big(\frac{n+1}{2n}\alpha^2 - \alpha\Big)(\bar{C}_i^2) - \frac{1}{n}\Big(\frac{n+1}{2n}\alpha^2\bar{C}_i - \alpha C_i\Big)(\sum_{j\in\mathcal{N}_i}\bar{C}_j)\Big] \qquad (2.23)$$

It now leaves to show that (2.23) is always non-positive, and only stays zero when the network is desynchronized. Let us denote Δp_i as the ith term of Δp, corresponding to the ith change value, such that

$$\Delta p = \sum_{i=1}^{N}\Delta p_i. \qquad (2.24)$$

Let us arrange the Δp_i terms into two groups: 1) Set $\mathcal{M}$ corresponding to when C_i is non-positive, and 2) Set $\mathcal{S}$ when it is positive.

1. If $C_i \leq 0$, then, from (2.12), $\bar{C}_i = 0$, and $\Delta p_i \in \mathcal{M}$ can be simplified as follows:

$$\Delta p_i = \frac{\alpha}{n}C_i\Big(\sum_{j\in\mathcal{N}_i}\bar{C}_j\Big) \qquad (2.25)$$

   Eqn. (2.25) is non-positive for any positive α, and the sum of all of the $\Delta p_i \in \mathcal{M}$ will also be non-positive.

2. If $C_i > 0$, then, from (2.12), $C_i = \bar{C}_i$, and the sum of all $\Delta p_i \in \mathcal{S}$ can be simplified as follows:

$$\sum_{i\in\mathcal{S}}\Delta p_i = \Big(\frac{n+1}{2n}\alpha^2 - \alpha\Big)\sum_{i\in\mathcal{S}}\Big[\bar{C}_i^2 - \frac{1}{n}\Big(\sum_{j\in\mathcal{N}_i}\bar{C}_i\bar{C}_j\Big)\Big] \qquad (2.26)$$

For the summation term in brackets, there will be $|\mathcal{S}| = s$ square terms and at most $\frac{sn}{n} = s$ non-zero pairwise terms, causing the expression to be strictly non-negative. Thus, (2.26) will be non-positive for any $\alpha \in (0, \frac{2n}{n+1})$.

Therefore, the sum of non-positive values results in the fact that (2.23) is always non-positive for any $\alpha \in (0, \frac{2n}{n+1})$.

We therefore must deal with the cases where Δp is zero and desynchronization has not yet been achieved. There are two such cases: 1) All change values are equal and positive ($C_1 = C_2 = \ldots = C_N > 0$), and 2) all change values are non-positive ($C_i \leq 0 \ \forall i$).

1. It is trivial to show that the first case cannot happen. If all change values are positive, then $\theta_i - \theta_{i-1} < \theta_{i+1} - \theta_i < \ldots < \theta_{i-1} - \theta_{i-2} < \theta_i - \theta_{i-1}$, which is a contradiction.

2. The second case can be true for one cycle since the nodes can sense only the "stale" phase from each node. However, this case will not be true for the next cycle. If no nodes move in a cycle, each node will then be observing the true phase of its connecting nodes, and not the stale values, in the next cycle. According to Lemma 2, $\sum_{i=1}^{N} C_i = 0$ holds in the next cycle. If desynchronization has not yet been achieved (i.e., not all change values are zero), then at least one node will have a positive change value, and will move forward in phase, causing the change in energy to be negative according to (2.23).

Therefore, the energy will always decrease and converge to zero. According to Proposition 2, the network will converge to the state of desynchronization. □

Remark 6. *As determined from the derivation of (2.23), a choice of change ratio $\alpha \in (0, \frac{2n}{n+1})$ will ensure that the network energy will decrease with each cycle. An α larger than $\frac{2n}{n+1}$ cannot be used to guarantee that the network energy will converge to zero, and thus cause the network to desynchronize. Because we want to choose α such that it can guarantee desynchronization independent of network size, the largest range of α, then, that can ensure desynchronization is $\alpha \in (0, 1]$, since $\min_{n \in \mathbb{N}} \frac{2n}{n+1} = 1$.*

2.3 Circulant Symmetric Case

We now relax the constraint of an all-to-all topology. We will show that any circulant symmetric network will desynchronize under our algorithm, specifically for the circulant symmetric

ring topology.

2.3.1 Convergence of the Ring Topology

Let us now consider the case of the circulant symmetric ring topology, where every node is connected to the node directly ahead of and behind it in phase.

Theorem 2. *A network with the circulant symmetric ring topology, where every node is connected to the node directly ahead of and behind it in phase, under the proposed algorithm will converge to the desynchronized state.*

Proof. We can prove Theorem 2 if we can show that $\Delta p(k) < 0$ for a general k. The proof follows a similar approach to Theorem 1.

Using the result from (2.16) in Theorem 1, and knowing that $n = 2$ for the ring topology, we can write an expression for the change value during the next cycle as follows:

$$C_i(k+1) = C_i(k) + \alpha\Big(\frac{\bar{C}_{i-1}(k) + \bar{C}_{i+1}(k)}{2} - \bar{C}_i(k)\Big) \tag{2.27}$$

Squaring both sides, subtracting $C_i(k)^2$ from both sides, and using the fact that $\bar{C}_i(k)^2 = C_i(k)\bar{C}_i(k)$ give us the term in the summation of (2.7):

$$\begin{aligned} &C_i(k+1)^2 - C_i(k)^2 \\ &\quad = \alpha\Big(C_i(k)\bar{C}_{i-1}(k) + C_i(k)\bar{C}_{i+1}(k) - 2\bar{C}_i(k)^2\Big) + \frac{\alpha^2}{4}\Big(\bar{C}_{i-1}(k) + \bar{C}_{i+1}(k) - 2\bar{C}_i(k)\Big)^2 \end{aligned} \tag{2.28}$$

Substituting (2.28) into (2.7) and then splitting the summation give us the following relationship, where we drop the cycle (k) notation for clarity:

$$\Delta p = \frac{1}{2}\Big(\alpha\sum_{i=1}^{N}\big(C_i\bar{C}_{i-1} + C_i\bar{C}_{i+1} - 2\bar{C}_i^2\big) + \frac{\alpha^2}{4}\sum_{i=1}^{N}\big(\bar{C}_{i-1} + \bar{C}_{i+1} - 2\bar{C}_i\big)^2\Big) \tag{2.29}$$

Through expanding and recombining, the first summation term in (2.29) can be simplified, resulting in a convenient shifting of indices as follows:

$$\alpha\sum_{i=1}^{N}\big(C_i\bar{C}_{i-1} + C_i\bar{C}_{i+1} - 2\bar{C}_i^2\big) = -2\alpha\Big(\sum_{i=1}^{N}\big[\bar{C}_i^2 - \frac{1}{2}(C_i\bar{C}_{i+1} + C_{i+1}\bar{C}_i)\big]\Big) \tag{2.30}$$

In a similar way, we can expand and recombine the second summation term in (2.29), also resulting in a convenient shifting of indices:

$$\frac{\alpha^2}{4}\sum_{i=1}^{N}\left(\bar{C}_{i-1}+\bar{C}_{i+1}-2\bar{C}_i\right)^2$$
$$=\frac{\alpha^2}{4}\sum_{i=1}^{N}\left(\bar{C}_{i-1}^2+\bar{C}_{i+1}^2+4\bar{C}_i^2-4\bar{C}_i\bar{C}_{i-1}-4\bar{C}_i\bar{C}_{i+1}+2\bar{C}_{i-1}\bar{C}_{i+1}\right)$$
$$=\frac{\alpha^2}{4}\Big(6\sum_{i=1}^{N}[\bar{C}_i^2]-8\sum_{i=1}^{N}[\bar{C}_i\bar{C}_{i+1}]+2\sum_{i=1}^{N}[\bar{C}_i\bar{C}_{i+2}]\Big) \quad (2.31)$$

By adding zero in a clever way, we can further simplify (2.31) as follows:

$$\frac{\alpha^2}{4}\Big(6\sum_{i=1}^{N}[\bar{C}_i^2]-8\sum_{i=1}^{N}[\bar{C}_i\bar{C}_{i+1}]+2\sum_{i=1}^{N}[\bar{C}_i\bar{C}_{i+2}]\Big)+\frac{\alpha^2}{4}\Big(2\sum_{i=1}^{N}[\bar{C}_i^2]-2\sum_{i=1}^{N}[\bar{C}_i^2]\Big)$$
$$=2\alpha^2\Big(\sum_{i=1}^{N}[\bar{C}_i^2-\bar{C}_i\bar{C}_{i+1}]\Big)-\frac{\alpha^2}{2}\Big(\sum_{i=1}^{N}[\bar{C}_i^2-\bar{C}_i\bar{C}_{i+2}]\Big) \quad (2.32)$$

We can now substitute (2.30) and (2.32) into (2.29):

$$\Delta p=\Bigg(-\alpha\Big[\sum_{i=1}^{N}\big(\bar{C}_i^2-\frac{1}{2}[C_i\bar{C}_{i+1}+C_{i+1}\bar{C}_i]\big)\Big]$$
$$+\alpha^2\Big(\sum_{i=1}^{N}[\bar{C}_i^2-\bar{C}_i\bar{C}_{i+1}]\Big)-\frac{\alpha^2}{4}\Big(\sum_{i=1}^{N}[\bar{C}_i^2-\bar{C}_i\bar{C}_{i+2}]\Big)\Bigg) \quad (2.33)$$

Further simplification gives us the following final result:

$$\Delta p=(\alpha^2-\alpha)\Big[\sum_{i=1}^{N}(\bar{C}_i^2)\Big]-\frac{\alpha^2}{4}\Big[\sum_{i=1}^{N}(\bar{C}_i^2-\bar{C}_i\bar{C}_{i+2})\Big]$$
$$-\sum_{i=1}^{N}\Big(\alpha^2[\bar{C}_i\bar{C}_{i+1}]-\alpha\Big[\frac{1}{2}(C_i\bar{C}_{i+1}+C_{i+1}\bar{C}_i)\Big]\Big) \quad (2.34)$$

It now leaves to show that (2.34) is always non-positive. We first notice that the first term is non-positive for $\alpha \in (0,1)$, and the second term is non-positive for any positive α. These results now leave to show that the third term causes the entire expression to always be non-positive.

We will consider the following three cases separately: 1) Both C_i and C_{i+1} are positive, 2)

one is positive, and the other non-positive, and 3) both are non-positive.

1. In the first case, the terms in brackets are equivalent, allowing us to combine the first and third term of (2.34):

$$(\alpha^2 - \alpha)\Big[\sum_{i=1}^{N}(\bar{C}_i^2)\Big] - \sum_{i=1}^{N}\big(\alpha^2[\bar{C}_i\bar{C}_{i+1}] - \alpha[\bar{C}_i\bar{C}_{i+1}]\big) = (\alpha^2 - \alpha)\Big[\sum_{i=1}^{N}(\bar{C}_i^2 - \bar{C}_i\bar{C}_{i+1})\Big] \quad (2.35)$$

   Eqn. (2.35) is non-positive for $\alpha \in (0, 1)$.

2. In the second case, the first term in brackets is zero, and the second term in brackets is non-positive. Thus, the whole term will be non-positive for any positive α.

3. In the third case, both terms in brackets are zero, making the whole term zero, and thus non-positive.

We next deal with the cases where Δp is zero and desynchronization has not yet been achieved. There are two such cases: 1) All change values are equal and positive ($C_1 = C_2 = \ldots = C_N > 0$), and 2) all change values are non-positive ($C_i \leq 0\ \forall i$). These are the same cases as for the all-to-all topology.

1. It is trivial to show that the first case cannot happen. If all change values are positive, then $\theta_i - \theta_{i-1} < \theta_{i+1} - \theta_i < \ldots < \theta_{i-1} - \theta_{i-2} < \theta_i - \theta_{i-1}$, which is a contradiction.

2. The second case can be true for one cycle since the nodes can sense only the "stale" phase from each node. However, this case will not be true for the next cycle. If no nodes move in a cycle, each node will then be observing the true phase of its connecting nodes, and not the stale values, in the next cycle. According to Lemma 2, $\sum_{i=1}^{N} C_i = 0$ holds in the next cycle. If desynchronization has not yet been achieved (i.e., not all change values are zero), then at least one node will have a positive change value, and will move forward in phase, causing the change in energy to be negative according to (2.34).

Therefore, the energy will always decrease and converge to zero for a choice of $\alpha \in (0, 1]$. According to Proposition 2, the network will converge to the desynchronized state. □

2.3.2 Convergence of Circulant Symmetric Topologies

We can further prove the convergence of any network that can be described with an adjacency matrix that is both circulant and symmetric to the desynchronized state.

Theorem 3. *A network with any circulant symmetric topology under the proposed algorithm will converge to the desynchronized state.*

Proof. The proof follows an identical structure as the proof for Theorem 2, and hence will be omitted here, due to tedious bookkeeping. Given the associated adjacency matrix for the network, we can derive the change in energy after each cycle. It will be non-positive for $\alpha \in (0, 1]$, and cannot stay at zero unless the network is desynchronized. Thus, the network energy will approach zero, and, according to Proposition 2, the network will converge to the state of desynchronization. □

Remark 7. *The definition of circulant symmetric effectively limits the region of initial conditions under which convergence can be achieved. Enforcing this condition on the initial phases for large networks may deserve separate investigation.*

Remark 8. *Other types of topologies may also desynchronize under our algorithm, but we cannot prove that from the theory presented in this paper. One would need to show that the splay state is the only equilibrium point for the network under the proposed algorithm, and that the energy for that network topology decreases with each firing cycle.*

Chapter 3

Simulation Results

We will now simulate how the network evolves over time using our proposed algorithm. We offer comparisons between the all-to-all topology and circulant symmetric ring topology. The results of these simulations confirm our theoretical results from the previous sections. All simulations were done using MATLAB, with all PCOs having identical fundamental frequency, $\omega = 2\pi$, and identical period of one second.

3.1 Proposed Algorithm Results

We consider a network of six PCOs, with randomly chosen initial phases from the interval $[0, 2\pi]$. For the circulant symmetric ring topology, we order the initial phases first, and then create the network such that every node is connected to the node directly ahead of and behind it in phase.

The evolution of the phases under our algorithm in the all-to-all topology and the circulant symmetric ring topology is given in Fig. 3.1. To visualize the evolution of the network toward desynchronization more easily, we also measure the phase difference between adjacent nodes, which we define in the following manner:

$$\Delta\theta_i = (\theta_{i+1} - \theta_i) \mod 2\pi, \text{ where } \theta_{N+1} = \theta_1 \tag{3.1}$$

Fig. 3.2 shows the plot of this phase distance between adjacent nodes, and confirms that desynchronization is indeed achieved.

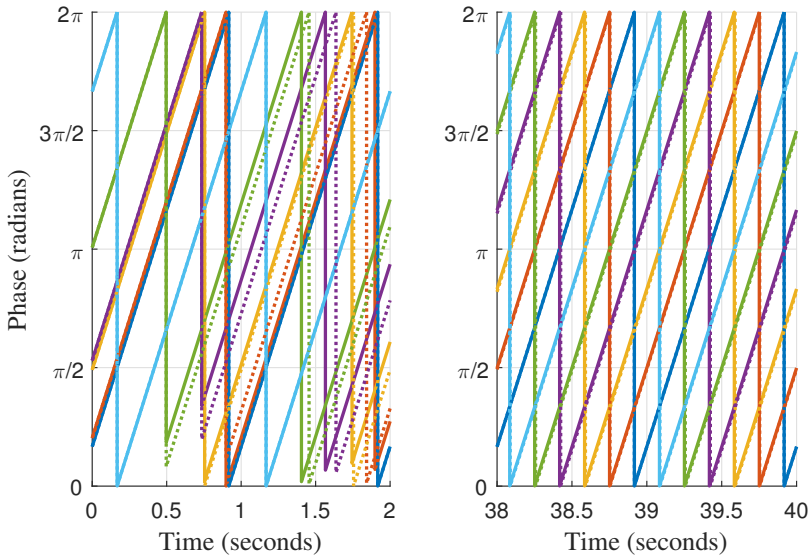


Figure 3.1: Initial and final phase evolution for six nodes with random initial state and $\alpha = 0.9$. Solid lines depict behavior under the all-to-all topology. Dotted lines depict behavior under the circulant symmetric ring topology. The network converges to the state of desynchronization.

In Fig. 3.3, we plot the change values for each node, which converge to zero with time. This behavior is to be expected, since once the network desynchronizes, no more jumps in phase need to be made.

In Fig. 3.4, the network energy as defined in (2.5) is shown for both the all-to-all and ring topology. It is important to note that the energy is not always strictly decreasing with regard to time. This result is primarily caused by the asymmetric nature of the update algorithm and secondarily due to the restriction that only forward jumps in phase can be made. However, the energy does strictly decrease while observing the value after every firing cycle, as was determined in our theoretical derivation. This result and our theoretical derivation show the importance of analyzing the evolution of the energy based on complete firing cycles instead of individual firing instants. Our treatment of desynchronization in this paper is different than the treatment of pulse-based desynchronization in the literature, which analyzes convergence to the splay state based on individual firing instances and will fail if applied directly to analyzing desynchronization in this paper.

As shown in Fig. 3.5, the rate at which the network energy converges is shown to be dependent on the network topology. More node connections cause the network to desynchronize more quickly than with fewer connections.

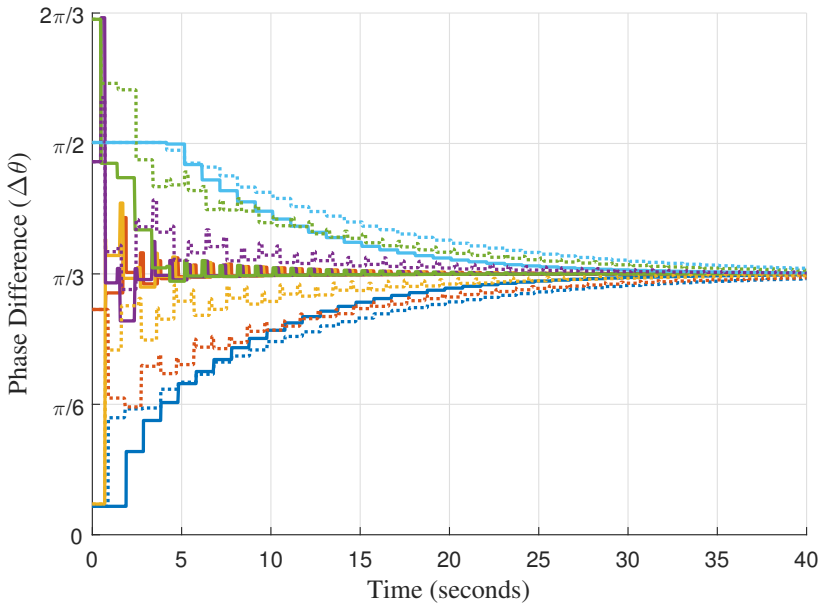


Figure 3.2: Phase differences for six nodes with random initial state and $\alpha = 0.9$. Solid lines depict values under the all-to-all topology. Dotted lines depict values under the circulant symmetric ring topology. The phase differences converge to the same value, one sixth of the unit circle distance, and all nodes become evenly spread out around the unit circle.

It is also important to note that the initial network energies under different network topologies are different. This difference is due to the increased number of connections in the all-to-all topology, making comparisons between the energy of the network and the closeness to desynchronization more difficult for networks of different topologies and sizes. Normalizing the energy curves to have the same initial value aids in this comparison.

3.2 Example of Insufficient Initial Condition

Although our proposed algorithm relaxes the constraint of an all-to-all network topology, our theory only expands the set of potential network topologies to circulant symmetric networks with ordered nodes. If the network does not satisfy the conditions of being circulant or symmetric when the nodes are ordered, then our theoretical results do not guarantee that the network will desynchronize.

As an example, we take the initial node phases from the previous example and swap the phases of node 1 and node 2, keeping the same network connections from before. The all-to-all topology, with the swapped nodes, still has a circulant symmetric adjacency matrix, and thus meets

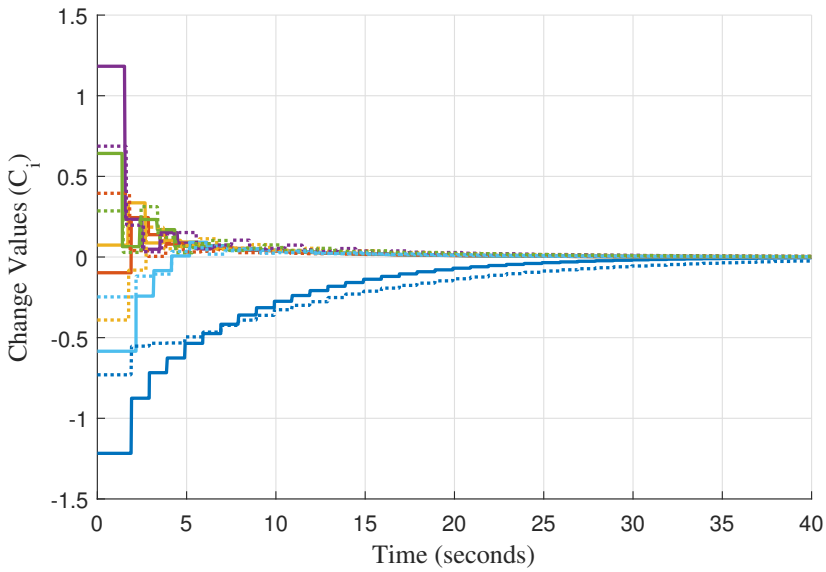


Figure 3.3: The evolution of the change value C_i for each node. Solid lines depict behavior under the all-to-all topology. Dotted lines depict behavior under the circulant symmetric ring topology. All change values converge to zero.

the conditions for our proposed algorithm. However, the resulting node ordering for the circulant symmetric ring topology causes the network to have a symmetric, but non-circulant, adjacency matrix.

As seen in Fig. 3.6, the nodes in the all-to-all topology achieve phase desynchronization. However, the nodes of the ring topology do not achieve phase desynchronization, but rather converge into groups of two, and do not maintain the initial firing sequence. Fig. 3.7 more apparently shows the effect of the ring topology not achieving desynchronization, and also shows the shifting of relative node positions.

Even though the network energy approaches zero, as shown in Fig. 3.8, the required initial conditions were not met for this topology, and the energy does not indicate desynchronization.

3.3 Comparison with DESYNC-STALE

Our proposed algorithm can be thought of as a variant and generalization of the DESYNC-STALE algorithm first given in [19]. The primary advantage that our proposed algorithm has over the DESYNC-STALE algorithm is the relaxation of the all-to-all topology constraint. Whereas the DESYNC-STALE algorithm assumes an all-to-all topology, our proposed algorithm can generalize to

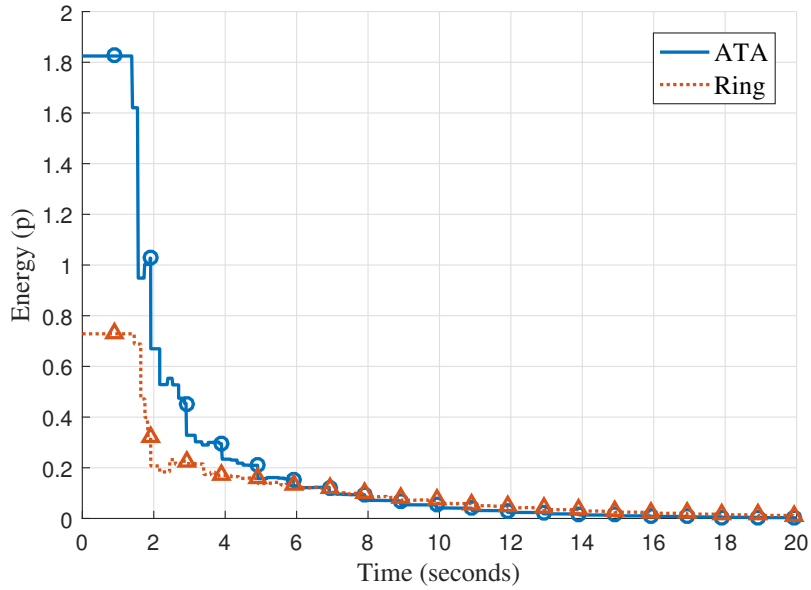


Figure 3.4: Network energy as a function of time for six nodes with random initial state and $\alpha = 0.9$ for both the all-to-all (ATA) and circulant symmetric ring topologies. Markers indicate the measured energy after a complete firing cycle. The energy converges to zero as the network desynchronizes.

any circulant symmetric network. Our proposed algorithm thus allows for a wider range of network topologies that can achieve desynchronization.

To illustrate the advantages of our proposed algorithm, we use the network topology shown in Fig. 2.3b, and use both algorithms to try to achieve the state of desynchronization. In the DESYNC-STALE algorithm, a node moves based on the nearest phase neighbor directly ahead of and behind it, whereas in our proposed algorithm the node moves based on all connected nodes.

In Fig. 3.9, we see that our proposed algorithm achieves evenly spaced node phases, whereas the DESYNC-STALE algorithm is unable to achieve desynchronization, as a result of each node ignoring the phase of one node to which it is connected.

Fig. 3.10 shows that our algorithm is able to accurately desynchronize under the given network topology, but the DESYNC-STALE algorithm is unable to accurately desynchronize. The desynchronization accuracy of the network $|\vec{\Delta}|$, defined in [19], is given to be "the sum of the absolute deviations from perfect desynchrony."

$$|\vec{\Delta}| = \sum_{i=1}^{N} \left| \Delta\theta_i - \frac{2\pi}{N} \right| \tag{3.2}$$

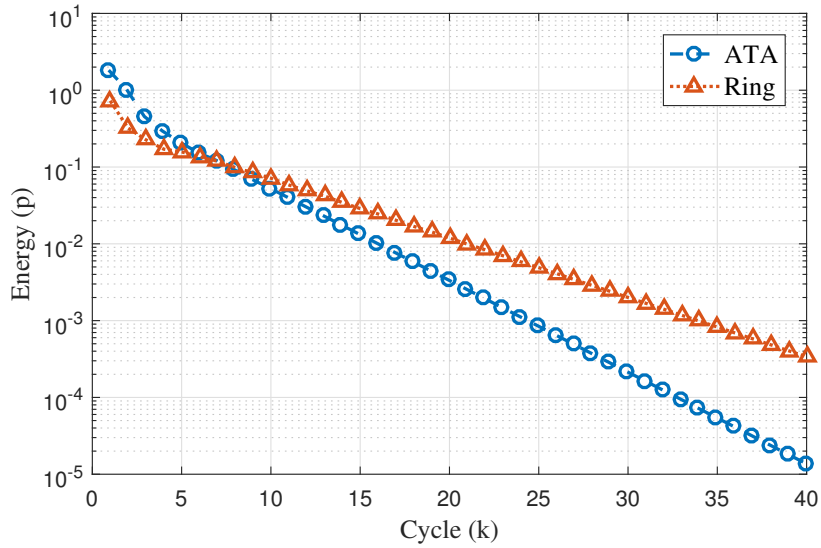


Figure 3.5: Network energy after a complete firing cycle for six nodes with random initial state and $\alpha = 0.9$ for both the all-to-all (ATA) and circulant symmetric ring topologies. The energy strictly decreases when measured after a complete firing cycle. The energy convergence rate depends on the network topology.

3.4 Proposed Algorithm Convergence

We now evaluate the speed of convergence of our algorithm for various sizes and types of topologies. To do so, we need some measure for when the network has desynchronized. We say that a network is ϵ-desynchronized after k cycles when the network energy satisfies $p(k) < \epsilon$.

In Fig. 3.11, we compare the relative convergence speed for various network topologies and sizes under different change ratios. We run 25 simulations at random initial conditions for each set of parameters, and plot the mean and standard deviation for when the network becomes ϵ-desynchronized. We see that more node connections cause the network to desynchronize more quickly for a given change ratio. Also, larger sizes of networks take more time to desynchronize. It is interesting to note that the ratio between the mean amount of cycles to ϵ-desynchronization for identical network size is approximately constant with regard to the change ratio.

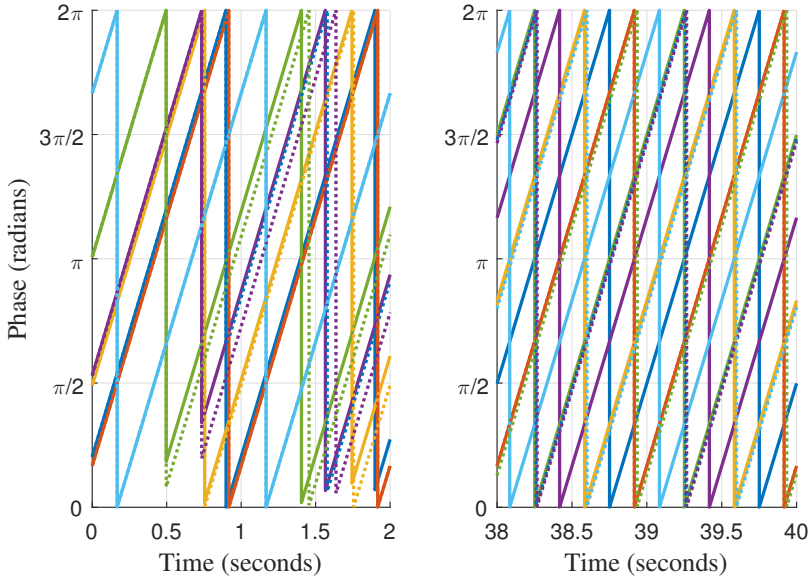


Figure 3.6: Initial and final phase evolution for six nodes with random initial state and $\alpha = 0.9$. Solid lines depict behavior under the all-to-all topology. Dotted lines depict behavior under the circulant symmetric ring topology. When unordered, the nodes in the circulant symmetric ring topology converge into groups of two, rather than spreading out evenly. The all-to-all topology is not affected by the reordering of the nodes.

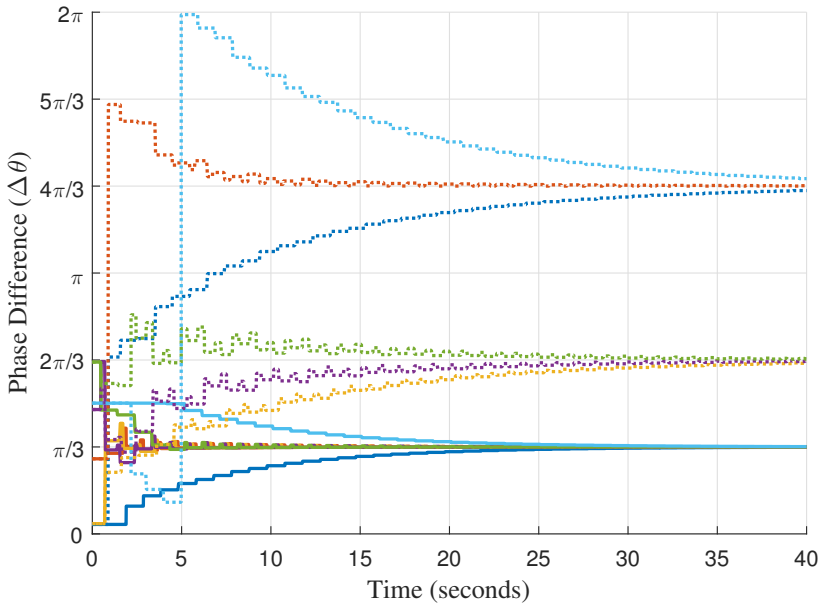


Figure 3.7: Phase differences for six nodes with random initial state and $\alpha = 0.9$. Solid lines depict values under the all-to-all topology. Dotted lines depict values under the circulant symmetric ring topology. The ring topology does not properly desynchronize or maintain the initial firing sequence when the nodes are unordered, and the resulting adjacency matrix is non-circulant.

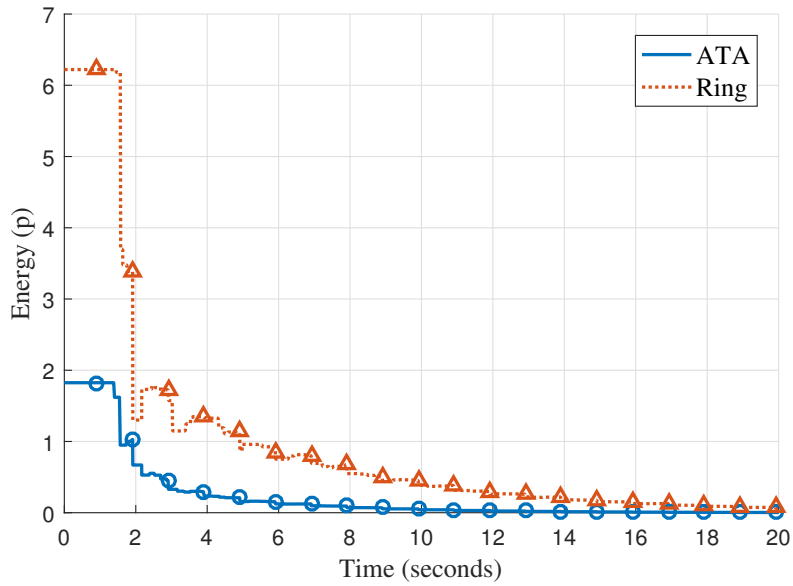


Figure 3.8: Network energy as a function of time for six nodes with random initial state and $\alpha = 0.9$ for both the all-to-all (ATA) and ring topologies. Markers indicate the measured energy after a complete firing cycle. The energy of the ring topology still approaches zero, but does not indicate desynchronization, since the ordered adjacency matrix is non-circulant.

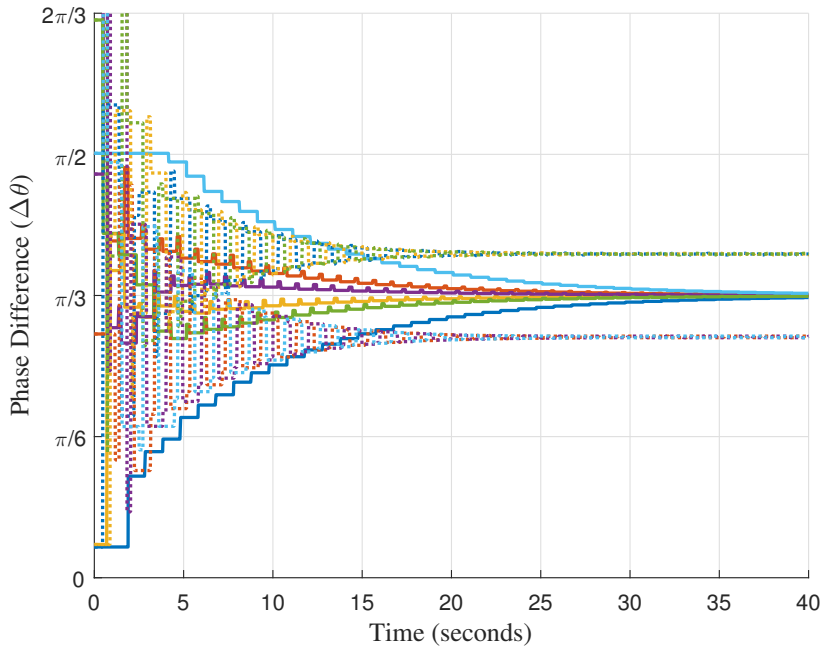


Figure 3.9: Phase Differences for six nodes with random initial state and $\alpha = 0.9$. Solid lines depict values under our proposed algorithm. Dotted lines depict values under the DESYNC-STALE algorithm. Both algorithms use the circulant symmetric network topology shown in Fig. 2.3b. The DESYNC-STALE algorithm is unable to completely desynchronize.

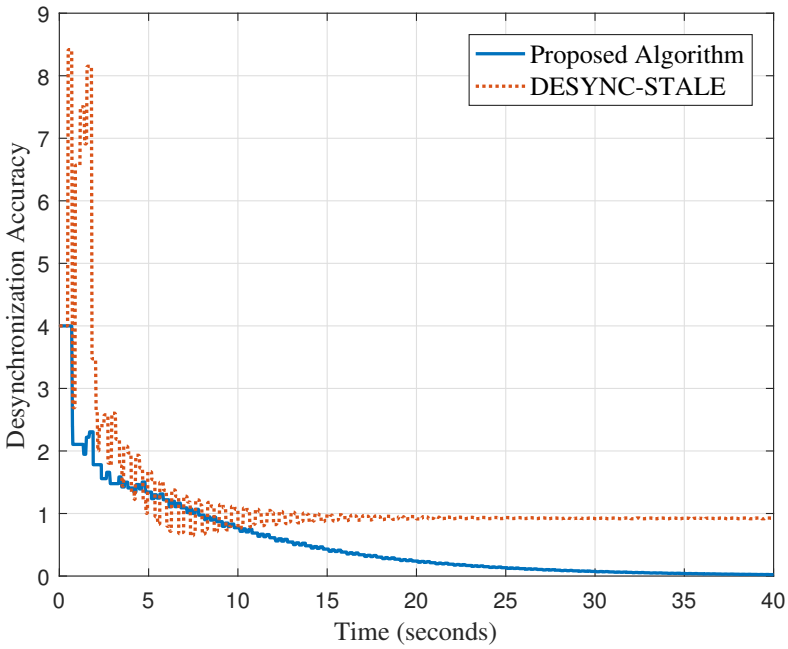


Figure 3.10: Desynchronization accuracy of the network as defined in [19] as a function of time for six nodes with random initial state and $\alpha = 0.9$. Solid lines depict behavior under our proposed algorithm. Dotted lines depict behavior under the DESYNC-STALE algorithm. The DESYNC-STALE algorithm fails to achieve network desynchronization.

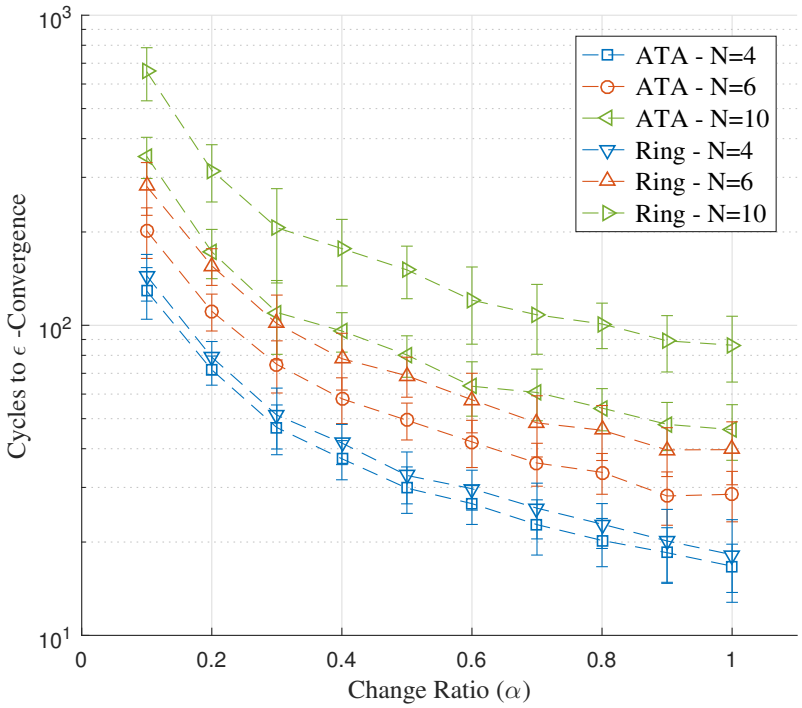


Figure 3.11: The convergence rate of the network toward desynchronization for both the all-to-all (ATA) and circulant symmetric ring topologies. Points represent the mean of 25 runs at random initial conditions for each change ratio to reach ϵ-desynchrony for $\epsilon = 1 * 10^{-4}$. Error bars show one standard deviation about the mean.

Chapter 4

Experimental Results

We now want to apply our algorithm to a real-world system, and demonstrate experimentally the proposed phase desynchronization algorithm. We seek to discover how a PCO network under the proposed algorithm behaves in more general conditions. Typical assumptions, such as negligible communication delay, identical oscillation period, perfect communication exchange, and instantaneous phase change, are not always valid in real systems. Deviations from these assumptions must be taken into account when designing a physical network with a pulse-coupled oscillator framework.

4.1 Roomba Robot Network

To demonstrate the proposed algorithm, we introduce Roomba robot platform to achieve coordinated motion control using the pulse-coupled oscillator model.

Roombas, as shown in Fig. 4.1, are small, circular robots that are primarily used to vacuum carpets in homes. However, these robots, with the vacuum components removed, can also be used as a programming testbed for engineers and hobbyists. By attaching a micro-controller to the input port of the Roomba, an external program can be run that sends commands to the robot according to the specifications given by the manufacturer. The micro-controller thus has direct access to the entire hardware of the robot, including the lights, motors, sensors, and speaker. Roombas are popular as a testbed system due to its simple and intuitive command scheme, its combination of sensors, and differential steering.

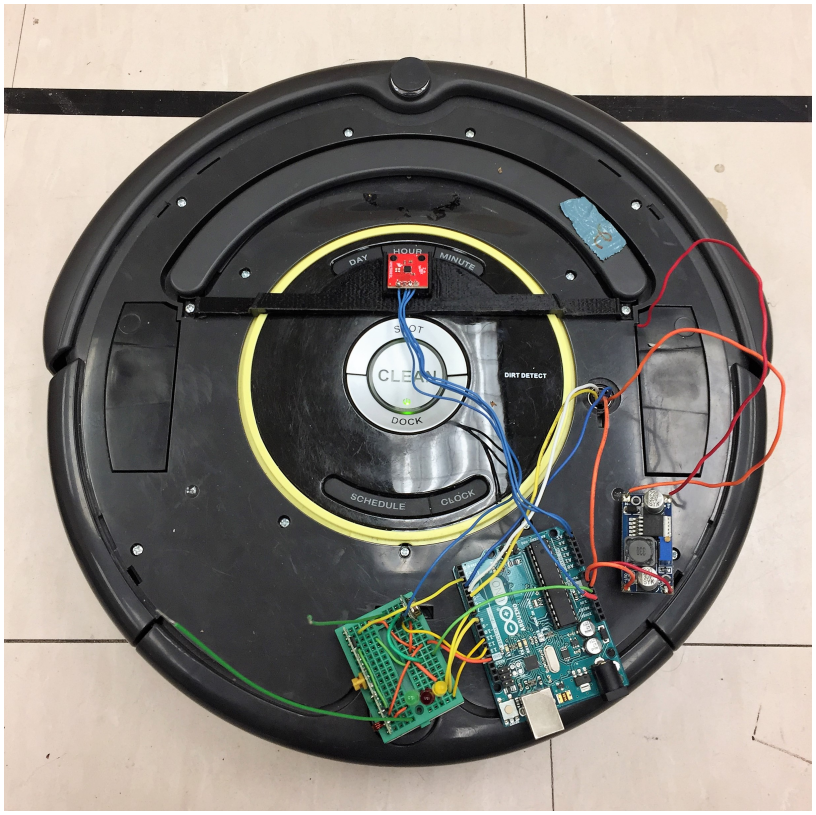
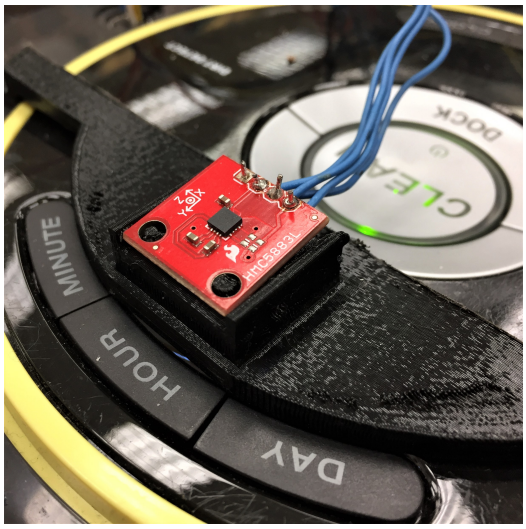


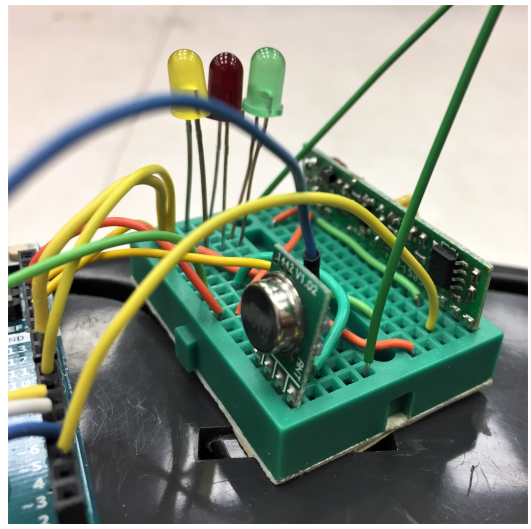
Figure 4.1: A single Roomba robot with the additional sensors needed to determine its orientation and communicate with other Roombas autonomously. It senses its orientation with a magnetometer, and communicates to other Roombas using a RF transmitter and receiver.

To control the Roombas, we utilize the Arduino Uno micro-controller. These small, inexpensive, boards have a simple computer interface and integrated development environment, on which programs can be written, compiled, and uploaded quickly in a custom programming language similar and compatible to the C and C++ programming languages. Its compatibility with C and C++ allow libraries written for those languages to be imported and used easily on Arduino. The many digital and analog inputs and outputs allow for easy and quick configuration for a variety of additional hardware.

In addition to the built in sensors on the Roomba and the Arduino micro-controller, we add on a magnetometer and a RF transmitter and receiver, as shown in Fig. 4.2. The magnetometer is a digital compass that measures the surrounding magnetic field in three dimensions, allowing for the calculation of local orientation. The RF transmitter and receiver give the Roombas a way to exchange information, or packets, directly with each other. For our purposes, the information



(a) HMC5883L magnetometer (digital compass) on a 3D-printed mount



(b) TWS-BS transmitter and RWS-371 receiver RF modules connected to an Arduino

Figure 4.2: External Roomba sensors to determine orientation and achieve pulse-coupled communication. The magnetometer is used to determine the Roomba's orientation. The RF modules are used to send pulse transmissions between Roombas.

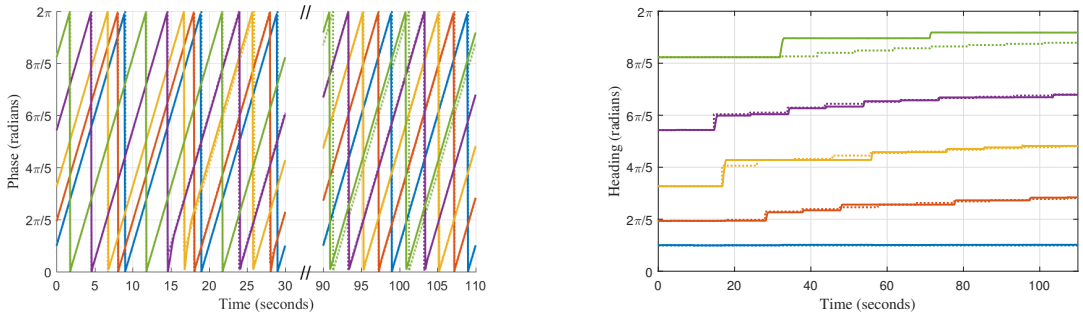
exchanged is a simple pulse, rather than a standard information packet.

With these additional components connected to the Roomba, we can now achieve the goal of having the Roombas desynchronize their headings by pointing in directions that are as different as possible. This goal is achieved by having each Roomba behave as an oscillator in a network under our proposed desynchronization algorithm, where the "phase" variable, θ, is the sum of the Roomba's heading, ϕ, and a continuously increasing counter, τ.

$$\theta = \phi + \tau$$

When a Roomba receives a pulse on its RF receiver, it will record the PRC value of its "phase" according to Eq. (2.1), and keep track of the total number of pulses that it heard from other Roombas between its own firings.

When the sum of the heading and the counter reaches the threshold value of 2π, the Roomba will send a pulse out over the RF transmitter. It will then update its heading value, ϕ, according to the phase update rule from Eq. (2.3) by rotating itself to have the magnetometer match the new heading value. Lastly, the Roomba will then subtract 2π from the counter value, τ. The combination of these steps is equivalent to an oscillator firing a pulse and updating its phase variable, θ, according



(a) Comparison of Roomba phase between simulation and experimental data

(b) Comparison of Roomba heading between simulation and experimental data

Figure 4.3: Phases and headings of 5 Roombas in an all-to-all topology under the proposed algorithm, with $\alpha = 0.9$, and cycle period of 10 seconds. Solid lines indicate the measured phase and heading of the Roomba. Dotted lines indicate the simulated phase and heading. The Roombas perform very closely to how the simulation predicts.

to the proposed algorithm.

Since the counter for each Roomba in the network is not directly adjusted according to the desynchronization algorithm, all of the counters will be equivalent across the network. Thus, over time, the headings of the Roomba will tend toward desynchronization, such that each Roomba is facing in as different a direction from the others as possible.

4.2 All-to-all Topology Results

To test out our proposed algorithm on the Roombas, we upload identical code onto each Arduino board that controls the Roomba, and gather data from the Arduino while observing the behavior of the Roombas. An example of the Arduino program used to implement the proposed algorithm is given in Appendix B. Given the initial conditions and design parameters of the network, we can also simulate in MATLAB how the Roombas should respond according to the desynchronization algorithm.

The first experiment we perform uses an all-to-all topology with each Roomba facing a random initial direction. The phase of the Roombas over the duration of the experiment is shown in Fig. 4.3a. As the figure shows, the phase of the Roombas matches well with the expected result of the simulation. To better see the performance of the Roombas, we also look at the headings of the Roombas, as shown in Fig. 4.3b, which should be separated as much as possible. We can see

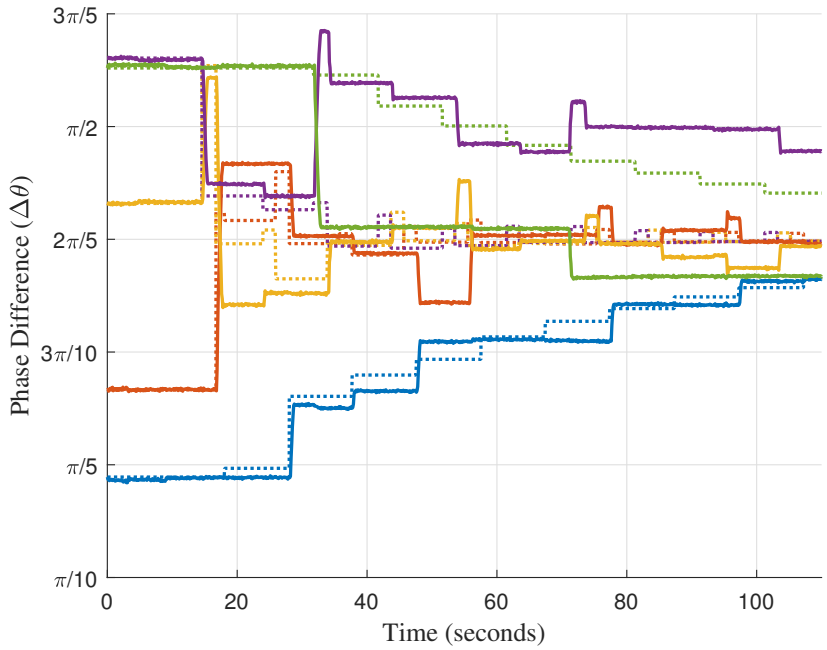


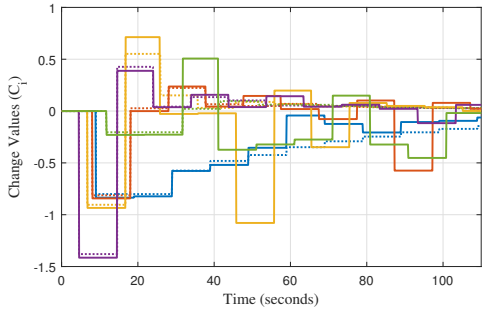
Figure 4.4: Phase differences for 5 Roombas in an all-to-all topology under the proposed algorithm, with $\alpha = 0.9$, and cycle period of 10 seconds. Solid lines indicate the measured phase differences between the Roombas. Dotted lines indicate the simulated phase differences. The Roombas are nearly able to achieve perfect desynchronization.

that the heading of the Roombas closely match the expected result of the simulation.

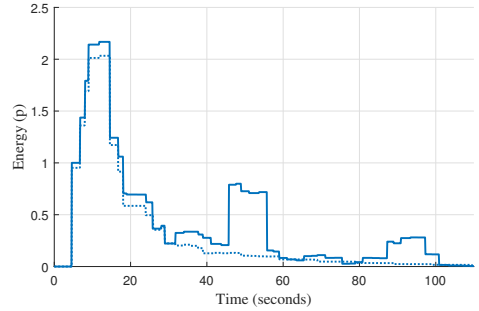
Another way to quantify the performance of the experiment is to view the phase differences between each Roomba, as given in Eq. (3.1). The phase differences between the Roombas are shown in Fig. 4.4. For a network of 5 Roombas, the phases should all be separated by an amount of $\frac{2\pi}{5}$. As the figure shows, the Roombas tend toward this amount of equal separation.

The final way to view the performance of the Roombas is to look at the change values and network energy. The change values, as shown in Fig. 4.5a, indicate the last amount that each Roomba determined to change, based on the timing of the received pulses during each cycle, and are calculated according to Eq. (2.2). The network energy, used in the analysis of Ch. 2 to prove convergence, is calculated according to Eq. (2.5), and shown in Fig. 4.5b. The change values and network energy do not always follow the simulated results as closely, but this behavior is primarily due to the Roombas not hearing every pulse in each cycle.

Remark 9. *An important note to make is that the network energy does not decrease during the first two cycles, but rather increases. This increase in network energy is due to the initialization of the*



(a) Comparison of Roomba change values between simulation and experimental data



(b) Comparison of Roomba network energy between simulation and experimental data

Figure 4.5: Change values and network energy of 5 Roombas in an all-to-all topology under the proposed algorithm, with $\alpha = 0.9$, and cycle period of 10 seconds. Solid lines indicate the measured change values and network energy of the Roombas. Dotted lines indicate the simulated change values and network energy.

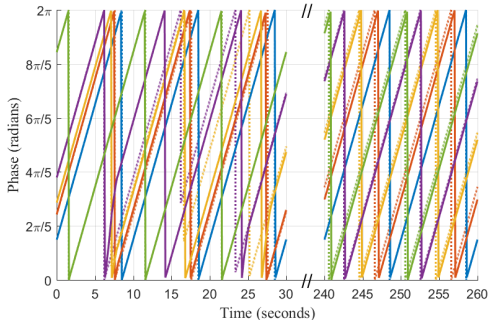
network, since the Roombas start out with no prior knowledge of the other Roombas in the network, and thus no amount of change occurs. However, during the second cycle, each Roomba is able to hear and respond to all of its neighboring Roombas, and thus the network energy begins to decrease as expected.

4.3 General Topology Results

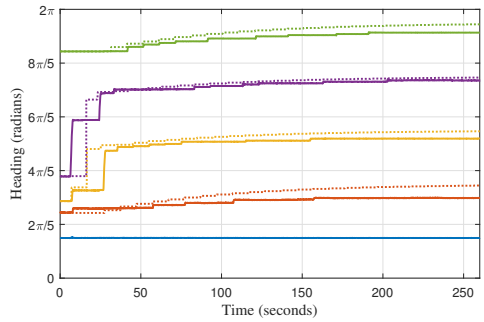
As the theory in Chapter 2 suggests, the network does not need to have an all-to-all topology to achieve desynchronization. To test this, we perform another experiment, assuming a more general topology of 5 Roombas, where each Roomba only hears and responds to the two Roombas farthest away from it in phase. We refer to this topology as a "star" topology, which is similar to a ring topology, where each Roomba responds to the two Roombas closest to it in phase. The star topology is the only other circulant symmetric topology possible with 5 nodes besides the ring topology and the all-to-all topology.

We first view the phase and heading of the network as it evolves. The phase of the Roombas are shown in Fig. 4.6a. To see more clearly the network's evolution, we also show the Roomba's heading in Fig. 4.6b. As can be seen, the network tends toward the state of desynchronization.

As in the all-to-all topology, the phase differences need to approach a constant value as the state of desynchronization is achieved. Fig. 4.7 shows the phase differences between the Roombas.



(a) Comparison of Roomba phase between simulation and experimental data



(b) Comparison of Roomba heading between simulation and experimental data

Figure 4.6: Phases and headings of 5 Roombas in a star topology under the proposed algorithm, with $\alpha = 0.9$, and cycle period of 10 seconds. Solid lines indicate the measured phase and heading of the Roomba. Dotted lines indicate the simulated phase and heading. The Roombas perform very closely to how the simulation predicts.

As shown, the phases, and thus the headings, of the Roombas tend toward the constant value of $\frac{2\pi}{5}$.

The final measure of desynchronization we use is the change values and network energy. The change values are shown in Fig. 4.8a, as well as the network energy in Fig. 4.8b. As in the all-to-all topology, these values are susceptible to pulse drops within each cycle, and thus cause the energy to deviate from the expected values.

4.4 Discussion

The Roombas do not perform exactly as the simulations indicate, and this behavior is due to numerous factors, besides the inherent variance and randomness of the measurement values.

The first and most significant deviation from the simulation is through pulse loss. The proposed algorithm assumes that all pulses are always received. However, the Roombas sometimes miss, or drop, a pulse that was sent. If a Roomba loses a pulse, it appears to that Roomba that one of the other Roombas has left the network, and thus will respond as if there is one fewer Roomba, even though the Roomba did not physically leave the network. The pulse loss causes the equilibrium of the desynchronized state to be upset, as if the Roomba had left and rejoined the network.

Additional error occurs due to the skew rate of the counters on each Roomba. In the Roomba network, the counter is based on the oscillation frequency of the internal clock on the Arduino board. However, even though this frequency is labeled as equivalent across all boards,

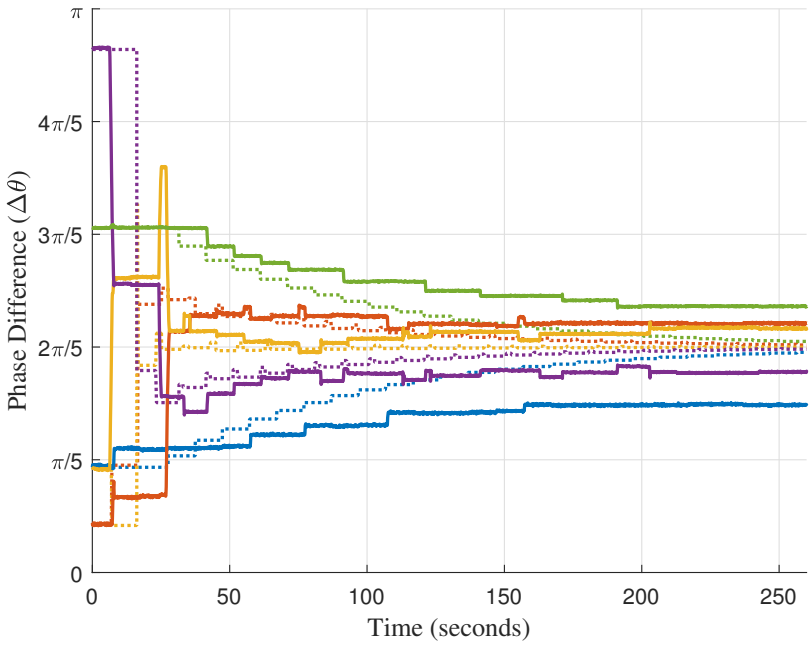
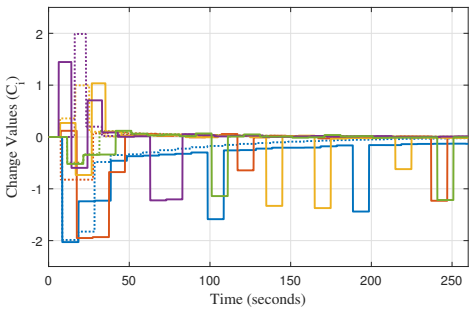


Figure 4.7: Phase differences for 5 Roombas in a star topology under the proposed algorithm, with $\alpha = 0.9$, and cycle period of 10 seconds. Solid lines indicate the measured phase differences between the Roombas. Dotted lines indicate the simulated phase differences. The Roombas are nearly able to achieve perfect desynchronization.

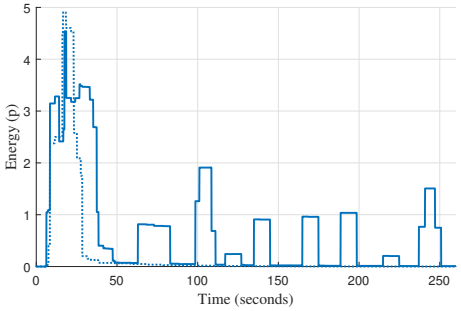
there are slight variations due to the manufacturing, causing the oscillation period of the Roombas to be slightly longer or shorter than the others. Since the phase correction is only applied to the heading, the counters will continue to diverge, unless the counters are manually reset after some time.

Some other minor sources of error do occur in our experiment. One is due to the small communication delay between the Roombas over the RF transmitter and receiver. Any small delay between when the pulse is sent and when the pulse is received (and enacted upon) will cause the phase to tend to decrease slightly. Another error can occur due to the continuity of the heading of the Roomba. Rather than updating its phase instantaneously, it takes time for the Roomba to adjust and turn its heading to the proper value. If a Roomba is in the middle of this adjustment when it receives a pulse, the value that the simulation would expect is going to be different than what the Roomba actually records, since it has not yet finished its heading adjustment.

However, even under these non-ideal conditions, the Roombas are still able to achieve a significant degree of desynchronization. The stability of the network is not strongly affected, and



(a) Comparison of Roomba change values between simulation and experimental data



(b) Comparison of Roomba network energy between simulation and experimental data

Figure 4.8: Change values and network energy of 5 Roombas in a star topology under the proposed algorithm, with $\alpha = 0.9$, and cycle period of 10 seconds. Solid lines indicate the measured change values and network energy of the Roombas. Dotted lines indicate the simulated change values and network energy.

the algorithm is robust to dynamic changes within the network. Furthermore, the algorithm is capable of achieving desynchronization in more general topologies than the completely connected network.

Chapter 5

Conclusion

In this thesis we present a novel control algorithm for achieving phase desynchronization in pulse-coupled oscillator networks. Our proposed algorithm is robust, allowing the control of the network to be independent of the number of nodes in the network. Having the control independent of the size of the network allows for the network to add and remove nodes freely without modifying the update strategy. This robustness presents itself useful in many applications, where the network may have nodes dynamically entering and leaving the network.

It is our belief that this thesis is the first to present a proof for the convergence to the desynchronized state for a pulse-coupled oscillator network topology other than the all-to-all network. Specifically, we show that the network, under our proposed dynamics, will converge to the state of uniform phase desynchronization for any circulant symmetric network with ordered nodes, which includes the all-to-all and circulant symmetric ring topologies as special cases. Relaxing the constraint of an all-to-all topology makes our proposed algorithm desirable for various applications, where direct communication may not be available between every pair of nodes in the network.

To verify our theory, we perform various computer simulations to demonstrate the behavior of the network under various topologies, as well as to compare the proposed algorithm with previous desynchronization algorithms. We also introduce the Roomba robot platform to create a pulse-coupled oscillator network to achieve coordinated motion control. We use the Roombas, along with the computer simulations, to analyze the proposed algorithm on a physical system. The results of the simulations and experiments show that the algorithm behaves as expected under a variety of circulant symmetric topologies.

Further analysis can still be made regarding the proposed algorithm. Specifically, theoretically analyzing the convergence speed of the network is desirable so that the network can be tuned by an appropriate choice of topology and change ratio. This analysis will be complex, since it will be dependent on the type of network topology and initial conditions of the nodes. It may also be desirable to more rigorously analyze the behavior of the network in the presence of pulse loss, frequency skew, and communication delay. As indicated in the experiments with the Roomba platform, pulse loss causes decreased performance in the network. Furthermore, research can be done to see if any additional types of network topologies can desynchronize under the proposed algorithm beyond the class of circulant symmetric networks.

Appendices

Appendix A Proofs of Lemmas

Proof for Lemma 1:

Proof. To prove this lemma, we need only to show that the firing sequence is invariant for one node, and then it will be true for all other nodes. To show the firing invariance for one node, we need only to show that the node will not pass the node directly ahead of it in phase within a firing cycle, regardless of whether the two nodes are connected.

Without loss of generality, let us consider node N, and show that it will not pass node 1 in phase during a firing cycle. We can describe this relationship mathematically with the following equation:

$$\alpha C_1 + \big((\theta_1 - \theta_N) \mod 2\pi\big) > \alpha C_N \tag{1}$$

We need only to consider the case when node N jumps forward in phase ($C_N > 0$), since the node will not pass the node directly in front of it if it does not jump. Without loss of generality, let us assume that the network has evolved to the instant right before node N fires ($\theta_N = 2\pi$). Thus, (1) can be rewritten as follows:

$$(\alpha C_1 + \theta_1) > \alpha C_N \tag{2}$$

We can express the amount that node N will jump when it fires as a function of the phases connected to it,

$$C_N = \frac{1}{n} \sum_{l \in \mathcal{N}_N} \big(- \big(2\pi - \big((\theta_l - \theta_N) \mod 2\pi\big)\big) + \pi\big) = \frac{1}{n} \sum_{l \in \mathcal{N}_N} \big(\theta_l\big) - \pi \tag{3}$$

where $l \in \mathcal{N}_N$ corresponds to the node that caused the mth instance of node N's recorded phase.

Similarly, we can express the amount that node 1 will jump,

$$C_1 = \frac{1}{n} \sum_{j \in \mathcal{N}_1} \big(- \big(2\pi - \big((\theta_j - \theta_1) \mod 2\pi\big)\big) + \pi\big) = \frac{1}{n} \sum_{j \in \mathcal{N}_1} \big(\theta_j\big) - \theta_1 - \pi \tag{4}$$

where $j \in \mathcal{N}_1$ corresponds to the node that caused the mth instance of node 1's recorded phase.

Since the nodes are ordered and the network is circulant, node 1 is connected to the nodes that are directly ahead in phase of the nodes that are connected to node N. Thus, we can express

(4) in terms of the neighbors of node N:

$$C_1 = \frac{1}{n} \sum_{l \in \mathcal{N}_N} \big(\theta_{l+1}\big) - \theta_1 - \pi \tag{5}$$

By substituting (3) and (5) into (2), we achieve the following relationship:

$$\alpha\Big(\frac{1}{n} \sum_{l \in \mathcal{N}_N} \big(\theta_{l+1}\big) - \theta_1 - \pi\Big) + \theta_1 > \alpha\Big(\frac{1}{n} \sum_{l \in \mathcal{N}_N} \big(\theta_l\big) - \pi\Big)$$
$$\implies \frac{\alpha}{n} \sum_{l \in \mathcal{N}_N} \big(\theta_{l+1}\big) + (1-\alpha)\theta_1 > \frac{\alpha}{n} \sum_{l \in \mathcal{N}_N} \theta_l \tag{6}$$

Since the nodes are ordered, then we know that $\theta_{l+1} > \theta_l \ \forall l \in \mathcal{N}_N$. Thus, for any $\alpha \in (0, 1]$, the expression in (6) will be true, and after a cycle the relative node positions and the resulting firing sequence will be maintained.

An issue arises if $\alpha C_N > \theta_1$ holds when node N fires for the first time. When this condition is true, then, due to the asynchronous update of the algorithm, node N will fire again before node 1, breaking the invariance of the firing sequence. Even though the amount that node 1 will change would put it past node N again, node N will fire first. However, this problem does not occur in practice where the nodes begin with no knowledge of the positions of neighboring nodes in the network. Each node senses the location of neighboring nodes based on received pulses as the network evolves. Specifically, node N has the largest initial phase and will fire first. When it fires, it will not have received any pulses from its neighboring nodes, and thus it has $C_N = 0$, and will not jump. As the network evolves, node $(N-1)$ will fire next. When it fires, it will only have received a pulse from node N if $N \in \mathcal{N}_{N-1}$, and, from the phase response function in (2.1), any jump that node $(N-1)$ makes will not break invariance. A similar result occurs for node $(N-2)$ to node 2. When node 1 fires, it will have full knowledge of the positions of all neighboring nodes, and from the analysis above, will jump and maintain firing invariance. Thus, when node N fires the second time, Eq. (2) holds, and invariance is maintained.

As an example, suppose that there are four nodes in an all-to-all topological network. Suppose the initial phases are given by the ordered phase vector $\vec{\theta} = [0, \frac{13\pi}{8}, \frac{7\pi}{4}, \frac{15\pi}{8}]^T$. If the nodes begin with initial knowledge of the location of the other nodes, then when node 4 fires, it has $C_4 = \frac{\pi}{4}$, and for any $\alpha > \frac{1}{2}$, node 4 will jump past node 1, and invariance is broken. However, because nodes

begin with no prior knowledge of neighboring nodes, when node 4 fires first, it has not received any pulses yet and hence its change value, C_4, is zero, which results in no jump. When node 3 fires, it has $C_3 = -\frac{7\pi}{8}$, and no jump occurs. Similarly, when node 2 fires, it has $C_2 = -\frac{13\pi}{16}$, and no jump occurs. Now, when node 1 fires, it has $C_1 = \frac{3\pi}{4}$, and will jump by amount αC_1. At the end of the first cycle, the resulting ordered phase vector is $\vec{\theta} = [\alpha\frac{3\pi}{4}, \frac{13\pi}{8}, \frac{7\pi}{4}, \frac{15\pi}{8}]^T$. When node 4 fires again during the second cycle, it has $C_4 = \frac{\pi}{4}$, and since the amount it will jump, $\alpha\frac{\pi}{4}$, is less than the distance to node 1, $(\frac{\pi}{8} + \alpha\frac{3\pi}{4})$, invariance is maintained. □

Proof for Lemma 2:

Proof. From (2.1) and (2.2), we can write that

$$n_i \sum_{i=1}^{N} C_i = \sum_{i=1}^{N} \sum_{m=1}^{n_i} Q(\theta_{i,m}) = \sum_{i=1}^{N} \sum_{l \in \mathcal{N}_i} \big(- [(\theta_i - \theta_l) \mod 2\pi] + \pi \big) \tag{7}$$

where $l \in \mathcal{N}_i$ corresponds to the node that caused the mth instance of node i's recorded phase.

Since the network is symmetric, then for two nodes j and k with a connection between them (assuming $j > k$), node j will record the m_1^{th} instance of its phase (denoted θ_{j,m_1}) when node k fires, and node k will record the m_2^{th} instance of its phase (denoted θ_{k,m_2}) when node j fires. Note that θ_{j,m_1} represents the phase difference between nodes k and j, and θ_{k,m_2} represents the phase difference between nodes j and k. Since no nodes have jumped in phase, the phase difference measured by these two nodes is the true phase difference between them and is equivalent for both nodes. Combining these two terms causes them to cancel.

$$Q(\theta_{j,m_1}) + Q(\theta_{k,m_2}) = \big(- (\theta_j - \theta_k) + \pi \big) + \big(- (\theta_k - \theta_j + 2\pi) + \pi \big) = 0 \tag{8}$$

Since the entire network is symmetric, and no nodes have jumped in the cycle, the whole expression can be recombined and canceled.

$$n_i \sum_{i=1}^{N} C_i = 0 \implies \sum_{i=1}^{N} C_i = 0 \tag{9}$$

Thus, the desired result is achieved. □

Appendix B Arduino Desynchronization Code for Roombas

```
/*  DeSync030817.ino
 *  Heading Deynchronization Code
 *  Uses Phase Desync algorithm in [Anglea-Wang, 2017]
 *  Last Updated: 3/17/2017
*/

#include <SoftwareSerial.h>
#include <SPI.h>
#include "VirtualWire.h"
#include "Wire.h"
#include "compassCUCI.h"

#define address 0x1E //0011110b, I2C 7bit address of HMC5883

const int rxPin = 3;            // Communication links to Roomba
const int txPin = 4;
const int ddPin = 5;
const int greenPin = 7;         // On when the Roomba sends a pulse.
                                //(very fast, may not see)
const int redPin = 8;           // On when Roomba heading update is 0
const int yellowPin = 11;       // On when the Roomba is turning.
const int transmit_pin = 12;    // Communication links to RF transmitter
const int receive_pin = 2;      // and receiver

/* Global variables for transmission and receiving */
unsigned char buf[VW_MAX_MESSAGE_LEN];
uint8_t buflen = VW_MAX_MESSAGE_LEN;
unsigned char message = 0;
```

```
/* Global variables needed to implement turn functions */
float angle;                        // Heading of Roomba
                                    // (found from digital compass)
float counter;                      // Global counter for Roomba (works
                                    // with angle to compute "phase")
float d_angle;                      // Change in angle that Roomba
                                    // will turn (updates each cycle)
float myPhase[6] = {0,0,0,0,0,0}; // Phase values at firing instances
float change = 0;                   // Change value of Roomba
int totalnodes = 0;                 // # of pulses received in a cycle
float DesiredHeading;               // Heading set point of Roomba;
int forward = 0;                    // Speed (mm/s) that Roomba wheels
                                    // turn to move forward;
                                    // must be in range [-400,400]
const float pi = 3.1415926;         // Pi to 7 decimal places

/* Adjustable Synchronization Parameters */
const float RATIO = 0.9;            // Ratio for amount to turn
                                    // must be in range (0 1]
const float EPSILON = 1.0;          // (ideally) Smallest resolution of
                                    // digital compass
boolean DHFlag = false;             // Desired Heading function indicator
                                    // (was the last command not to turn?)

/* Sync Counter Setup */
unsigned long millisCounter;        // Base time for determining counter
const float counterspeed = 36;      // Number of times per second that the
                                    // counter increments - range (0,1000]
                                    // Adjust this value to vary the
                                    // speed/frequency of the counter
```

```
const float millisRatio = counterspeed / 1000;
                                    // Counter increments per millisecond
const float millisAdjust = 360000 / counterspeed;
                                    // Amount of counter adjustment/firing
const unsigned long counterAdjust = (unsigned long) millisAdjust;
                                    // Truncate to unsigned long

/* Data Parameters */
const int dataTIMER = 100; // Number of milliseconds between data points

/* Roomba Serial Setup */
SoftwareSerial Roomba(rxPin, txPin); // Set up Roomba communication

/* Data Point Collector Setup */
unsigned long deltime;
unsigned long resettime;
long sno = 0;

/* Start up Roomba */
void setup() {
  Serial.begin(115200);          // Declare Serial Monitor baud rate
  display_Running_Sketch();      // Show sketch information at startup
  Serial.println("Loading...");
  pinMode(ddPin, OUTPUT);        // Set the pins as outputs
  pinMode(greenPin, OUTPUT);
  pinMode(redPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  Roomba.begin(115200);          // Declare Roomba baud rate
  digitalWrite(greenPin, HIGH); // Say we're alive
```

```
// set up ROI to receive commands
Roomba.write(byte(7));      // RESTART
delay(8000);
Serial.print("STARTING ROOMBA.");
Roomba.write(byte(128));   // START
delay(50);
Roomba.write(byte(131));   // CONTROL
//131 - Safe Mode
//132 - Full mode (Be ready to catch it!)

// Test to make sure Roomba communication is working
Roomba.write(byte(139));   // Turn on Dirt Detect light
Roomba.write(byte(25));    // and Green Clean button
Roomba.write(byte(0));
Roomba.write(byte(128));
delay(500);
Roomba.write(byte(139));   // Change Green to Red
Roomba.write(byte(25));
Roomba.write(byte(255));
Roomba.write(byte(128));
delay(500);
Roomba.write(byte(139));   // Turn off Clean Button
Roomba.write(byte(25));
Roomba.write(byte(255));
Roomba.write(byte(0));
delay(50);

/* Transmitter Setup */
Wire.begin();       //Initialize Serial and I2C communications
```

```
/* Put the HMC5883 IC into the correct operating mode */
Wire.beginTransmission(address);  //open communication with HMC5883
Wire.write(0x02);                 //select mode register
Wire.write(0x00);                 //continuous measurement mode
Wire.endTransmission();

/* Compass Calibration: */
compass_init(1);      // Set Compass Gain
Move(0, -75);         // Set roomba spinning to calibrate the compass
                      // Spins ~2.75 rotations CCW.
compass_debug = 0;    // Show Debug Code in Serial Monitor
                      // (Set to 0 to hide Debug Code)
compass_offset_calibration(2); // Find compass axis offsets
Move(0, 0);           // Stop spinning after completing calibration

/* Initialise the IO and ISR */
vw_set_tx_pin(transmit_pin);
vw_set_rx_pin(receive_pin);
vw_set_ptt_inverted(true);  // Required for DR3100
vw_setup(2000);             // Bits per second

/* Receiver Setup */
delay(1000);
Serial.print(" Setup");

vw_rx_start();              // Start the receiver PLL running

digitalWrite(yellowPin, HIGH);
Serial.println("...complete");
delay(500);
```

```
  digitalWrite(yellowPin, LOW);

  digitalWrite(greenPin, LOW);  // say we've finished setup
  /* Wait for command to initialize synchronization */
  delay(1000);
  angle = Calculate_Heading();  // Throw away first calculation
  delay(200);
  /* Initialize synchronization */
  angle = Calculate_Heading();  // Determine initial heading information
  sendPalse();                  // Send reset Palse
  resetCounters();              // Reset counter values
}

void loop() { // Swarm "Heading Synchronizaiton" Code
  /* Read angle from compass */
  angle = Calculate_Heading();     // Set angle from the compass reading
  counter = millisRatio * (long)(millis() - millisCounter);
                                   // Find current counter value
  if (angle + counter < 0) {  // If the "phase" is negative...
    millisCounter = millisCounter - counterAdjust;// Adjust base counter
  } // Happens when oscillator fires and moves heading past 360 degrees.
  /* Receive a pulse signal */
  recievePulse();

  /* Send a pulse signal and adjust heading */
  if (angle + counter >= 360) { // If the "phase" reaches 360 degrees...
    sendPulse();                                    // Fire pulse
    millisCounter = millisCounter + counterAdjust;// Adjust base counter
    // Adjust DesiredHeading
    if (totalnodes > 0) {   // If I've heard at least one oscillator
```

```
    PhaseUpdateF(); // Update Roomba Heading (only forward motion)
    totalnodes = 0; // Clear value for next cycle.
  } // Else ignore if no nodes were heard
} // Ignore if the angle and counter are less than 360 degrees.

/* Receive a pulse signal */
recievePulse();

/* If a pulse signal was recieved */
if (message == 'b') { // If reset palse is received...
  //Serial.println(" Reset Palse.");// Include for debugging
  digitalWrite(redPin, HIGH);      // Notify that we received palse
  digitalWrite(greenPin, HIGH);
  resetCounters();                 // Reset the counter variables
  digitalWrite(redPin, LOW);       // End Notify that we received palse
  digitalWrite(greenPin, LOW);
  message = 0; // Clear the message variable
} else if (message == 'z') {
  //Serial.println(" Sync Pulse."); // Include for debugging
  digitalWrite(yellowPin, HIGH);   // Notify that we received pulse
  myPhase[totalnodes] = (angle + counter); // Record current phase
  totalnodes++;                    // Increment total number of nodes
  digitalWrite(yellowPin, LOW);    // End Notify that we received pulse
  message = 0; // Clear the message variable
}

/* Receive a pulse signal */
recievePulse();

/* Turn to the DesiredHeading set point */
```

```
  DH_Turn();

  /* Send a data point to the Serial monitor */
  if (millis() - deltime >= dataTIMER) { // Every so often...
    deltime += dataTIMER;    // Reset base value for data points
    sno++;                   // Increment the data point number
    Serial.println(";");     // End row, start new row of data
    Print_Heading_Data();    // Print data point to the Serial Monitor
  }

  /* Receive a pulse signal */
  recievePulse();

  /* Reset Counters of all Roombas every 5 minutes */
  if (millis() - resettime >= 300000) { // If it's been 5 minutes...
    sendPalse();     // Send Reset Palse
    resetCounters();// Reset Counter values
  }
}/* Go back and check everything again. Should be fast */

/* SUBROUTINES */

/* Recieve pulse from RF transmitter and save value */
void recievePulse() {
  if (vw_get_message(buf, &buflen)) {  // If I receive a pulse ...
    message = (char)buf[0]; // Return pulse character
  }
}

/* Sends out a pulse when phase equals 360 degrees */
```

```
void sendPulse() {
  char pulse[2] = {'z'};
  //Serial.println("Sync Pulse Sent.");      // Include for debugging
  digitalWrite(greenPin, HIGH); // Tell me that I'm sending a pulse
  vw_send((uint8_t *)pulse, strlen(pulse));
  vw_wait_tx();                   // Wait until the whole message is gone
  digitalWrite(greenPin, LOW);  // Tell me that I'm done sending a pulse
}

/* Sends out a palse when new Roomba finishes setup */
void sendPalse() {
  char palse[2] = {'b'};
  //Serial.println("Reset Pulse Sent.");     // Include for debugging
  digitalWrite(greenPin, HIGH); // Tell me that I'm sending a palse
  digitalWrite(redPin, HIGH);
  vw_send((uint8_t *)palse, strlen(palse));
  vw_wait_tx();                   // Wait until the whole message is gone
  digitalWrite(greenPin, LOW);  // Tell me that I'm done sending a palse
  digitalWrite(redPin, LOW);
}

/* Reset ALL the counters */
void resetCounters() {
  millisCounter = millis();    // Reset base counter (on all robots)
  deltime = millis();          // Reset base value for data output
  resettime = millis();        // Reset base value for reset timer
  sno = 0;                     // Reset data point counter
  DesiredHeading = angle;      // Reset heading set point
  counter = 0;                 // Reset counter value
  change = 0;                  // Reset change value
```

```
  totalnodes = 0;              // Reset total number of nodes
  Serial.println();            // Move to next line
  Print_Heading_Data();        // Display initial heading information
}

/* Update heading due to received pulses (only adjust forward) */
void PhaseUpdateF(void) {
  change = 0;    // Clear out the variable
  for(int i = 0; i < totalnodes; i++) {
    // Accumulate total change value
    change = change + (180 - myPhase[i])/totalnodes;
    myPhase[i] = 0; // Clear the phase value after it is used.
  } // change will not be modified until the next heading update firing

  if (change > 0) { // Don't change if amount is negative
    // Update desired heading
    DesiredHeading = DesiredHeading + (RATIO*change);
    if (DesiredHeading < 0) {        // Normalize the heading value
      DesiredHeading += 360;
    } else if (DesiredHeading >= 360) {
      DesiredHeading -= 360;
    }
    digitalWrite(redPin, LOW); // Indicate a change in heading
  } else {
    digitalWrite(redPin, HIGH); // Indicate no change in heading
  }
}

/* Set Roomba spin to acheive the value of DesiredHeading
 * DesiredHeading is the set point for the heading
```

```
 * EPSILON is desirably small (probably in range [0.5, 1])
*/
void DH_Turn(void) {
  if (angle < (DesiredHeading + EPSILON) &&
      angle > (DesiredHeading - EPSILON) && DHFlag == false) {
    // If it's not moving, and I'm close enough...
    digitalWrite(yellowPin, LOW); // Say we have stopped turning.
    return;                       // Leave function
  }

  int spinValue; // Speed of wheels in mm/s to spin
  float holder;
  float thresh1 = 25; // First threshold value
  float thresh2 = 5;  // Second threshold value
  holder = angle - DesiredHeading;
  holder = abs(holder);   // Absolute difference of where I am (angle)
                          // and where I want to be (DesiredHeading)
  // Determine spin speed based on Thresholds
  if (holder > thresh1 && holder < (360 - thresh1)) {
    spinValue = 100;  // Move faster
  } else if (holder > thresh2 && holder < (360 - thresh2)) {
    spinValue = 50;   // Move fast
  } else { // if (holder <= thresh2)
    spinValue = 15;   // Move slow (keeps down oscillations
  }                   // due to main loop execution rate)

  // Determine direction of spin
  if(DesiredHeading < EPSILON) {
    if(angle > (DesiredHeading + EPSILON) &&
       angle < (DesiredHeading + 180) ) {
```

```
      // Spin Left (CCW)
      Move(forward, -spinValue);
      DHFlag = true;
    } else if ((angle < (360 + DesiredHeading - EPSILON)) &&
              (angle >= (DesiredHeading + 180)) ) {
      // Spin Right (CW)
      Move(forward, spinValue);
      DHFlag = true;
    } else {
      // Stop Spinning
      Move(forward, 0);
      DHFlag = false;
      digitalWrite(yellowPin, LOW); // Say we have stopped turning.
    }
  } else if(DesiredHeading < 180) {
    if(angle > (DesiredHeading + EPSILON) &&
       angle < (DesiredHeading + 180) ) {
      // Spin Left (CCW)
      Move(forward, -spinValue);
      DHFlag = true;
    } else if ((angle < (DesiredHeading - EPSILON)) ||
              (angle >= (DesiredHeading + 180)) ) {
      // Spin Right (CW)
      Move(forward, spinValue);
      DHFlag = true;
    } else {
      // Stop Spinning
      Move(forward, 0);
      DHFlag = false;
      digitalWrite(yellowPin, LOW); // Say we have stopped turning.
```

```
    }
  } else if(DesiredHeading < (360 - EPSILON)) {
    if ((angle < (DesiredHeading - EPSILON)) &&
        (angle > (DesiredHeading - 180)) ) {
      // Spin Right (CW)
      Move(forward, spinValue);
      DHFlag = true;
    } else if ((angle > (DesiredHeading + EPSILON)) ||
              (angle <= (DesiredHeading - 180)) ) {
      // Spin Left (CCW)
      Move(forward, -spinValue);
      DHFlag = true;
    } else {
      // Stop Spinning
      Move(forward, 0);
      DHFlag = false;
      digitalWrite(yellowPin, LOW); // Say we have stopped turning.
    }
  } else {
    if ((angle < (DesiredHeading - EPSILON)) &&
        (angle > (DesiredHeading - 180)) ) {
      // Spin Right (CW)
      Move(forward, spinValue);
      DHFlag = true;
    } else if ((angle > (DesiredHeading + EPSILON - 360)) &&
              (angle <= (DesiredHeading - 180)) ) {
      // Spin Left (CCW)
      Move(forward, -spinValue);
      DHFlag = true;
    } else {
```

```
        // Stop Spinning
        Move(forward, 0);
        DHFlag = false;
        digitalWrite(yellowPin, LOW); // Say we have stopped turning.
      } // End "else angle"
    } // End "else DesiredHeading"
} // End Function

/* General Wheel Motor command function.
     X = common wheel speed (mm/s); Y = differential wheel speed;
     X > 0 -> forward motion; Y > 0 -> CW motion
     This function allows for both turning and forward motion.
     Error may result if |X|+|Y| > 500 (Max value is 500)
     */
void Move(int X, int Y) {
 unsigned int RW = (X - Y);
 unsigned int LW = (X + Y);
 Roomba.write(byte(145));
 Roomba.write(byte((RW & 0xff00) >> 8));
 Roomba.write(byte(RW & 0xff));
 Roomba.write(byte((LW & 0xff00) >> 8));
 Roomba.write(byte(LW & 0xff));
 return;
}

float Calculate_Heading(void) {
  float t;
  compass_scalled_reading(); // Get Raw data from the compass
  /* Heading Calculation (y-axis front) */
  t = (atan2(compass_y_scalled,compass_x_scalled) * (180/pi) - 90);
```

```
  if (t < 0) {
    t = t + 360;
  }
  return t;
}

/* Display Heading Information to the Serial Monitor */
void Print_Heading_Data(void) {
  Serial.print(sno);        // Data point number
  Serial.print(", ");
  Serial.print(angle);      // Robot Heading
  Serial.print(", ");
  Serial.print(counter);    // Counter value
  Serial.print(", ");
  Serial.print(DesiredHeading);       // Desired Set point
                                      // (angle should follow this)
  Serial.print(", ");
  Serial.print(change);     // Current change value
}

/* Displays the Sketch running on the Arduino.
 * Use at startup on all code. */
void display_Running_Sketch (void) {
  /* Find the necessary informaiton */
  String the_path = __FILE__;
  int slash_loc = the_path.lastIndexOf('\\');
  String the_cpp_name = the_path.substring(slash_loc + 1);
  /* Display to the Serial Monitor */
  Serial.print("\nSketch Name: ");
  Serial.println(the_cpp_name);
```

```
  Serial.print("Compiled on: ");
  Serial.print(__DATE__);
  Serial.print(" at ");
  Serial.print(__TIME__);
  Serial.println("\n");
}
```

Bibliography

[1] C. S. Peskin. *Mathematical aspects of heart physiology*. Courant Institute of Mathematical Science, New York University, New York, 1975.

[2] B. Ermentrout. Type I membranes, phase resetting curves, and synchrony. *Neural Computation*, 8(5):979–1001, 1996.

[3] R. Mirollo and S. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[4] P. Goel and B. Ermentrout. Synchrony, stability, and firing patterns in pulse-coupled oscillators. *Physica D: Nonlinear Phenomena*, 163(3):191–216, 2002.

[5] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, pages 142–153, New York, NY, USA, 2005. ACM.

[6] Y.W. Hong and A. Scaglione. A scalable synchronization protocol for large scale sensor networks and its applications. *IEEE Journal on Selected Areas in Communications*, 23(5):1085–1099, May 2005.

[7] Y. Q. Wang and F. J. Doyle III. Optimal phase response functions for fast pulse-coupled synchronization in wireless sensor networks. *IEEE Transactions on Signal Processing*, 60(10):5583–5588, Oct 2012.

[8] Y. Q. Wang, F. Núñez, and F. J. Doyle III. Energy-efficient pulse-coupled synchronization strategy design for wireless sensor networks through reduced idle listening. *IEEE Transactions on Signal Processing*, 60(10):5293–5306, Oct 2012.

[9] F. Dörfler and F. Bullo. Synchronization in complex networks of phase oscillators: A survey. *Automatica*, 50(6):1539 – 1564, 2014.

[10] J. Nishimura. Frequency adjustment and synchrony in networks of delayed pulse-coupled oscillators. *Physical Review E*, 91:012916, Jan 2015.

[11] F. Núñez, Y. Q. Wang, and F. J. Doyle III. Synchronization of pulse-coupled oscillators on (strongly) connected graphs. *IEEE Transactions on Automatic Control*, 60(6):1710–1715, June 2015.

[12] F. Núñez, Y. Q. Wang, and F. J. Doyle III. Global synchronization of pulse-coupled oscillators interacting on cycle graphs. *Automatica*, 52:202–209, 2015.

[13] A. V. Proskurnikov and M. Cao. Synchronization of pulse-coupled oscillators and clocks under minimal connectivity assumptions. *CoRR*, abs/1510.02338, 2015.

[14] R. Gentz, A. Scaglione, L. Ferrari, and Y. W. P. Hong. Pulsess: A pulse-coupled synchronization and scheduling protocol for clustered wireless sensor networks. *IEEE Internet of Things Journal*, PP(99):1–13, 2016.

[15] J. Klinglmayr, C. Bettstetter, M. Timme, and C. Kirst. Convergence of self-organizing pulse-coupled oscillator synchronization in dynamic networks. *IEEE Transactions on Automatic Control*, PP(99):1–1, 2016.

[16] G. Brandner, U. Schilcher, and C. Bettstetter. Firefly synchronization with phase rate equalization and its experimental analysis in wireless systems. *Computer Networks*, 97:74 – 87, 2016.

[17] R. Sepulchre, D. A. Paley, and N. E. Leonard. Stabilization of planar collective motion: All-to-all communication. *IEEE Transactions on Automatic Control*, 52(5):811–824, May 2007.

[18] H. Gao and Y. Q. Wang. Integrated communication and control for collective motion with limited communication. In *20th IFAC World Congress*, accepted, 2017.

[19] A. Patel, J. Degesys, and R. Nagpal. Desynchronization: The theory of self-organizing algorithms for round-robin scheduling. In *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pages 87–96, Cambridge, MA, July 2007.

[20] J. Degesys and R. Nagpal. Towards desynchronization of multi-hop topologies. In *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 129–138, Oct 2008.

[21] R. Pagliari, Y. W. P. Hong, and A. Scaglione. Bio-inspired algorithms for decentralized round-robin and proportional fair scheduling. *IEEE Journal on Selected Areas in Communications*, 28(4):564–575, May 2010.

[22] S. Phillips and R. G. Sanfelice. Results on the asymptotic stability properties of desynchronization in impulse-coupled oscillators. In *American Control Conference (ACC), 2013*, pages 3272–3277, Washington DC, USA, June 2013. IEEE.

[23] N. Deligiannis, J. F. C. Mota, G. Smart, and Y. Andreopoulos. Decentralized multichannel medium access control: Viewing desynchronization as a convex optimization method. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, IPSN '15, pages 13–24, New York, NY, USA, 2015. ACM.

[24] H. Gao and Y. Q. Wang. Analysis and design of phase desynchronization in pulse-coupled oscillators. http://arxiv.org/pdf/1603.03313v1.pdf.

[25] R. Sepulchre, D. A. Paley, and N. E. Leonard. Stabilization of planar collective motion with limited communication. *IEEE Transactions on Automatic Control*, 53(3):706–719, April 2008.

[26] P. Taechalertpaisarn, S. Choochaisri, and C. Intanagonwiwat. An orthodontics-inspired desynchronization algorithm for wireless sensor networks. In *2011 IEEE 13th International Conference on Communication Technology (ICCT)*, pages 631–636, Sept 2011.

[27] D. Buranapanichkit, N. Deligiannis, and Y. Andreopoulos. Convergence of desynchronization primitives in wireless sensor networks: A stochastic modeling approach. *IEEE Transactions on Signal Processing*, 63(1):221–233, Jan 2015.

[28] T. Anglea and Y. Q. Wang. Phase desynchronization: A new approach and theory using pulse-based interaction. *IEEE Transactions on Signal Processing*, 65(5):1160–1171, March 2017.

[29] H. K. Khalil. *Nonlinear Systems*. Pearson Education. Prentice Hall, 2002.