2022 Future Computing Summer Internship Project: Detecting if simulated oscillators are following the Kuramoto Model by determining if the oscillators experience synchronization.

Nicholas Schantz

August 4, 2022

Abstract

1 Project Description

The challenge addressed by this work is to determine a correct and efficient way to analyze multiple oscillators and determine which oscillators are synchronized and which are not. The purpose for this is to detect synchronization of oscillators in simulation. This problem is being looked at due to the Kuramoto Model, which is a mathematical model that describes how coupled oscillators may synchronize over time. If this behavior is unintended and system architects are unaware of it, this can create unintended behavior. In this case, an effort is put forward to detecting which oscillators are synchronized in a set and which are not.

2 Motivation

3 Prior work

Detecting synchronization in signals appears to be largely researched in the neuroscience field. Different methods for detecting synchronization have been researched. Biswas, Khamaru, and Majumdar research a method to determine a signal synchronization measurement based on measuring signal peaks [1]. Similarly, Quian, Kreuz, Grassberger, defined a method to analyzing signal synchronization through the use of defining events [2]. The authors use peaks as their event in which to analyze for signal synchronization. They measure for synchronization by comparing when events occur in multiple signals. Their idea to focus on peaks in signal was used in the implementation for oscillator synchronization detection in discrete-event simulators.

4 Result

Due to the Structural Simulation Toolkit (SST) discrete-event simulator's (DES) limitations, it was not feasible to implement the algorithm for detecting oscillator synchronization in the DES. This is due to component frequencies being set at the beginning of the simulation and unable to be changed or offset during runtime. In this case, the current implementation was written in Python and we will theorize how it can be transferred to a discrete-event simulator.

The current implementation for detecting oscillator synchronization is to sample when peaks occur in a set of signals during a sample of one period for all signals. This implementation assumes that the frequency of all signals are tracked and ignores signal amplitudes. Since frequency is given, we will assume that the set of oscillators being compared all share the same frequency. The goal now is to determine if they are all in phase or in anti-phase. The algorithm for synchronization detection works as follows:

1. During runtime, begin sampling the set of oscillators for one period. As they share the same frequency, one period is enough to determine if they are synchronized.

2. While sampling, when the first peak is encountered, store which oscillator it is associated with and the time it occurred. This will be the reference oscillator peak.
3. Continue sampling for the period and store each time the first peak has occurred for all other oscillators.
4. After sampling for one period, calculate the differences in time between the reference oscillator's first peak and all other oscillators' first peaks.
5. Multiply the differences in time by 2π to give the phase difference between the reference oscillator and all other oscillators.
6. Compare the phases of the oscillators with each other to determine if they are in-phase (phase difference of zero) or anti-phase (phase difference of π).

The psuedocode for the algorithms for detecting oscillator synchronization:

Algorithm 1 Oscillator Sampling

```
for length(Period) do
    if peaks detected == number of oscillators then
        Break
    end if
    for each signal do
        if current signal's first peak has already been found then
            Continue
        end if
        if signal's value == signal's amplitude then
            if reference peak found == false then
                Store current signal as reference oscillator
                Store current time as reference oscillator's first peak
                Reference peak found equals true
                Set current signals first peak found to be true
                Increment peaks detected
            else
                Store current signals time subtracted from the time of the reference oscillator's first peak
                multiplied by 2π
                Set current signals first peak found to be true
                Increment peaks detected
            end if
        end if
    end for
end for
```

Algorithm 2 Oscillator Synchronization Comparison

```
boiler
```

The algorithm was written in python and only tested on oscillators with simple harmonic motion such as sine and cosine function. It worked as intended but it has not been tested on more complicated oscillators.

5 Future Work

Due to time constraints, researching and coming up with an implementation for detecting oscillator synchronization was rushed. Due to this, the implementation explained above was only tested on oscillators with simple harmonic motion. The algorithm should be tested against more complicated oscillators that share the same frequencies to see if the metric is still accurate.

A different promising direction is to look for research related to phase synchronization, phase clustering, and phase correlation. These terms all relate to calculating the phases of signals and comparing them with other signals to extract information about the pair. Similar to the above, researching these measurements was cut short due to time constraints.

References

[1] Rahul Biswas, Koulik Khamaru, and Kaushik K. Majumdar. A peak synchronization measure for multiple signals. *IEEE Transactions on Signal Processing*, 62(17):4390–4398, 2014.

[2] R. Quian Quiroga, T. Kreuz, and P. Grassberger. Event synchronization: A simple and fast method to measure synchronicity and time delay patterns. *Phys. Rev. E*, 66:041904, Oct 2002.