Algorithms **2008**, *1*, 1-x; DOI: 10.3390/algor10x000x

OPEN ACCESS
algorithms
ISSN 1999-4893
www.mdpi.org/algorithms

Article

Neural Oscillators for Robotics Programming

Patrick McDowell*, Theresa Beaubouef

Department of Computer Science and Industrial Technology
Southeastern Louisiana University, SLU 10847
Hammond, LA 70402 USA
E-mail: pmcdowell@selu.edu, E-mail: tbeaubouef@selu.edu

* Author to whom correspondence should be addressed.

Received: / Accepted: / Published:

Abstract: Neural mechanisms such as central path generators have been discovered to control basic rhythmic movements in animals. In programming robotics applications, models of neural oscillators are used for control. Although there has been significant study in this area by biologists, it is difficult to find basic guidelines for the programming of neural oscillators by computer scientists. In this paper, the authors approach neural oscillators from a programmer's point of view, providing background and examples for developing systems involving neural oscillators for use in robotics applications.

Keywords: neural oscillator, central path generator, robotics, algorithm

---

1. Introduction

Scientists have studied neural oscillators for decades, and research in the biological processes involving neural oscillators and central path generators for various types of organisms continues. Significant work has been done in trying to understand the functions of various neurons and components of such biological neural networks [14, 15], developing properties general to all networks of neural oscillators [5], and the modeling of processes such as locomotion in simple animals [4, 6, 7, 10, 13]. Associative neural network models with behavior similar to central path generators have also been developed [9].

Computer scientists have researched various models and developed artificial neural networks that model natural neural networks to some extent. Artificial intelligence research into the machine learning technique based of neural networks is a well developed field, and the literature in this general area quite vast. Research related to neural oscillators and central pattern generators using these computerized neural network models, such as in [11] is not as prevalent. However, the use of such models in controlling the motion of robots has been well established [3, 8, 12]. For robotics researchers who are developing programs for robots and wanting to apply techniques of neural oscillators and/or central path generators for programming robotic control without the necessity of understanding complex theoretical mathematical models or learning about how simple invertebrates swim, there appears to be no single source of basic information available. This article provides a basic non-biological background on neural oscillators from a programmer's perspective and discusses the design and coding of neural oscillators.

2. Basic Neural Oscillator

The purpose of a neural oscillator is to generate a repeating output that ranges between a maximum and minimum value given constant input. This is different from other transformation functions, such as a neural network, because most transformations generate a unique output that is specific to each input. Said another way, a neural oscillator generates a unique pattern of outputs over time for each unique input set. To illustrate the point, think of the process of walking. A person may think "walk slowly", but what comes out of his brain is a set of rhythmic patterns that move the legs through a walking gait. To walk faster, one generally thinks "walk faster" and the part of the brain that controls walking increases the frequency and/or amplitude of the signals being sent to the leg muscles. Thus, the input "walk" produces a pattern of outputs that repeat over time that control the leg muscles.

Neural oscillators have two or more nodes connected by weights. In a typical two state oscillator, one state can be thought of as an Exciter state and the other as an Inhibitor state. Figure 1 below illustrates a typical two state oscillator.

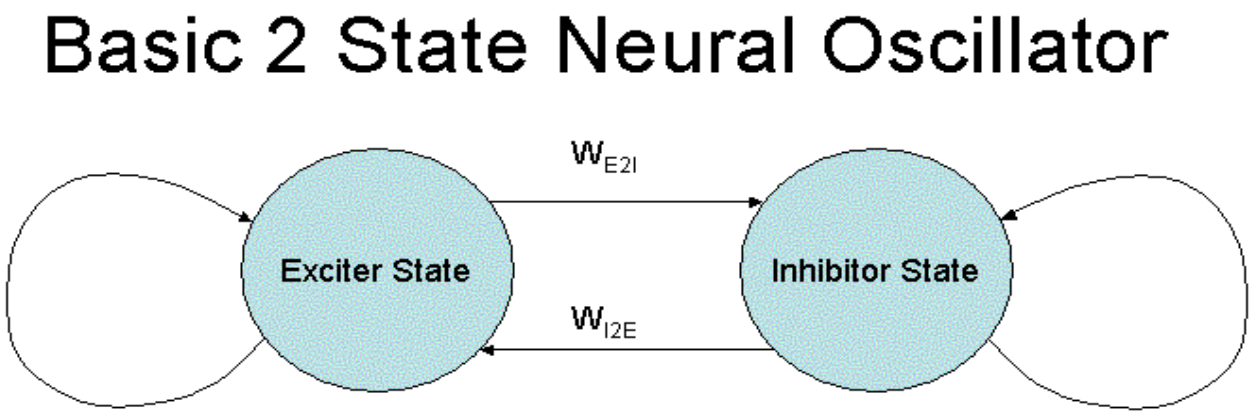

Figure 1. Basic 2 state oscillator. After the output of the system has reached the maximum value control moves to the Inhibitor state, after the final value is multiplied by the Exciter to Inhibitor weight W_{E2I}. The reverse happens in the Inhibitor state.

The general functioning of the oscillator is as follows. Starting the output value V_m at initial value V_{stable} in the Exciter state the oscillator circulates through the Exciter state adding to V_m until the oscillator's maximum value V_{max} is obtained. When this occurs, control transfers to the Inhibitor state with the value of V_m multiplied by the weight W_{E2I}. For now, assume that V_m is passed to the Inhibitor state unchanged, that is $W_{E2I} = 1$. Once in the Inhibitor state, control remains there, diminishing V_m until it reaches the oscillator's minimum value V_{min}. At that point control returns to the Exciter state, with V_m being multiplied by the weight W_{I2E}. As before, assume for now that $W_{I2E} = 1$. The pattern will repeat indefinitely as long as the weights connecting the Exciter and Inhibitor states remain 1. If they are less than 1, the pattern will eventually diminish; however, if the weights are greater than 1, the system will become unstable. It becomes unstable because each time control is passed from the Exciter state to the Inhibitor state, V_{max} grows by a factor of W_{E2I} and the converse is true for V_{min}.

One of the most basic patterns that can be generated is the saw-tooth pattern. It is created by adding a constant value to V_m in the Exciter state and subtracting a constant value from V_m in the Inhibitor state. Figure 2 below illustrates the pattern with adding a constant C = 1 to the pattern. Notice that the pattern is not diminishing, i.e., the amplitude remains constant, due to the inter-state weights being equal to 1. Figure 3, however, shows the result of making the Exciter to Inhibitor and Inhibitor to Exciter state weights less than 1. Here the pattern diminishes because W_{E2I} and W_{I2E} are less than one.

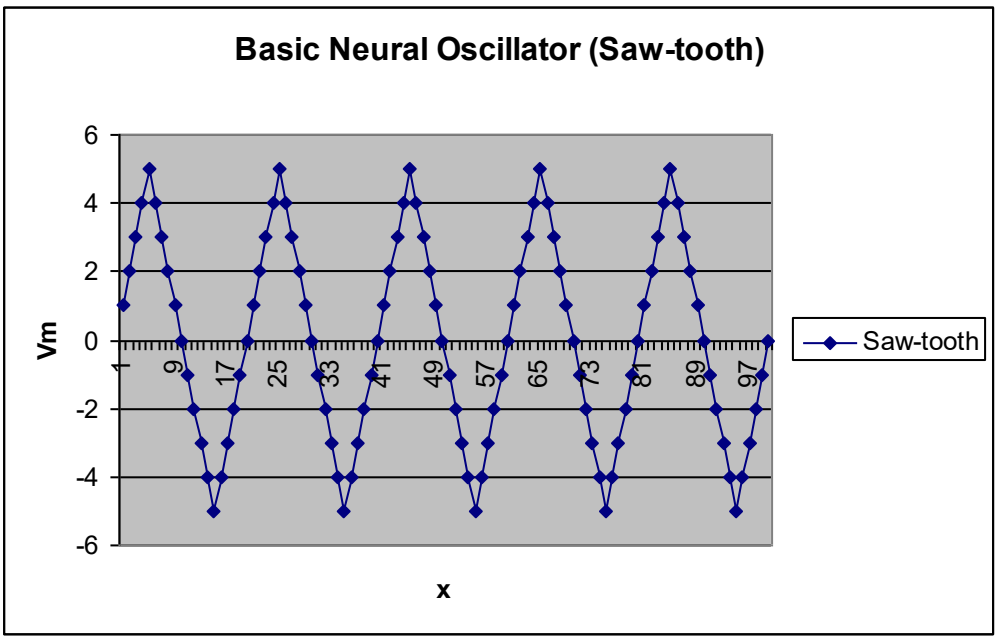

Figure 2. Output of a neural oscillator created by adding a constant (C = 1) in the Exciter state and subtracting a constant (C = 1) in the Inhibitor state.

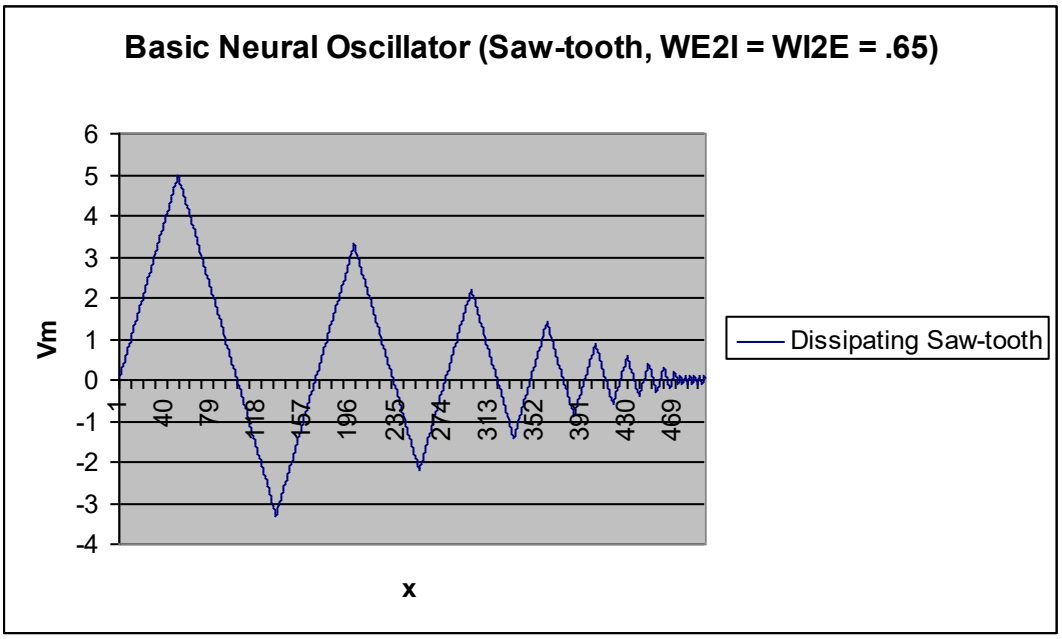

Figure 3. Output of a neural oscillator created by adding a constant (C = .1) in the Exciter state and subtracting a constant (C = .1) in the Inhibitor state with weights W_{E2I} and W_{I2E} less than 1.

A code snippet in C for the above example follows:

```
/* Initialize variables. */
j = 0;                          // j is the time variable (x-axis)
V_stable = (Vmax + Vmin)/2.0f;
Vm = V_stable;                  // Vm is the output
Vd_work = Vmax;                 // maximum value for Vm
V0_work = Vmin;                 // minimum value for Vm
state = 'e';                    // state ‘e’ is for Exciter, ‘i’ is for Inhibitor

while(running) {                // running is TRUE until system shutdown

        /* Excite state. */
        if (state == 'e') {

                Vm = Vm + 0.1f;         // here .1 is the constant C

                /* Get ready to go to inhibit state. */
                if (Vm >= (Vd_work - T)) {              // T is the tolerance value
                        state = 'i';
                        V0_work = V0_work * wE2I;       // wE2I is weight from Exciter to Inhibitor

                        /* Check for shutdown. */
                        if (V0_work > T) {
                                running = FALSE;
                        }
                }
        }

        /* Inhibit state. */
        else if (state == 'i') {
                Vm = Vm - 0.1f;         //here .1 is the constant C

                /* Get ready to go to excite state. */
                if (Vm <= (V0_work + T)) {              //T is the tolerance value
```

```
                        state = 'e';
                        Vd_work = Vd_work * wI2E;       //wI2E is weight from Inhibitor to Exciter

                        /* Check for shutdown. */
                        if (Vd_work < T) {
                                running = FALSE;
                        }
                }
        }

        /* Output data. */
        fprintf(out, "%f \n", Vm);

        /* Increment count. */
        j = j + 1;

        /* Check for shutdown. */
        if (j >= numDats) {                     // numDats is max number of time steps
                running = FALSE;
        }
}
```

While the saw-tooth pattern illustrates the arrangement of the states and connecting weights, the waveform it generates is not what is typically found to be desirable in the literature. Most of the waveforms shown are much more continuous in nature, many times appearing not too dissimilar from a sine wave. One of the most simple, semi-continuous wave forms that can be generated is the "shark fin" wave form, as can be seen in Figure 4. This waveform is created by using a proportional controller [15] in the Exciter state to achieve the maximum desired value, and then once this value is reached, the control is handed off to the Inhibiter state, which in turn uses a proportional controller to diminish the value to the minimum desired value.

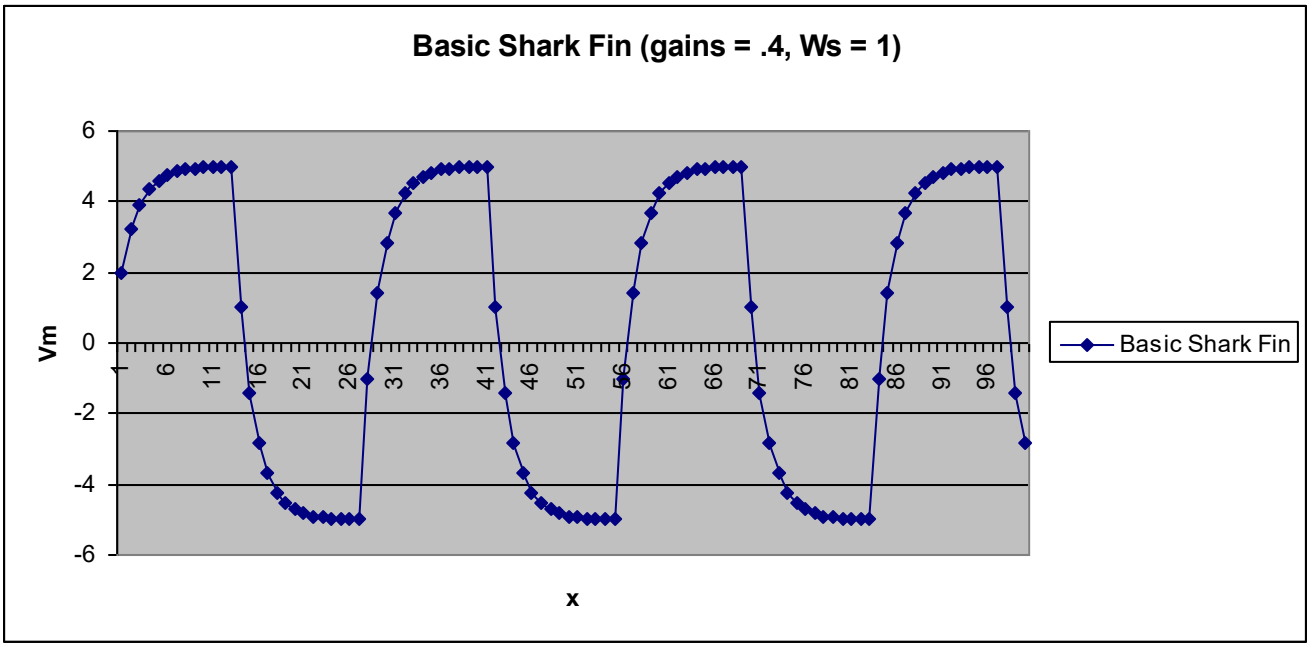

Figure 4. Output of the basic Shark Fin neural oscillator. Here the waveform is continuous and non-diminishing.

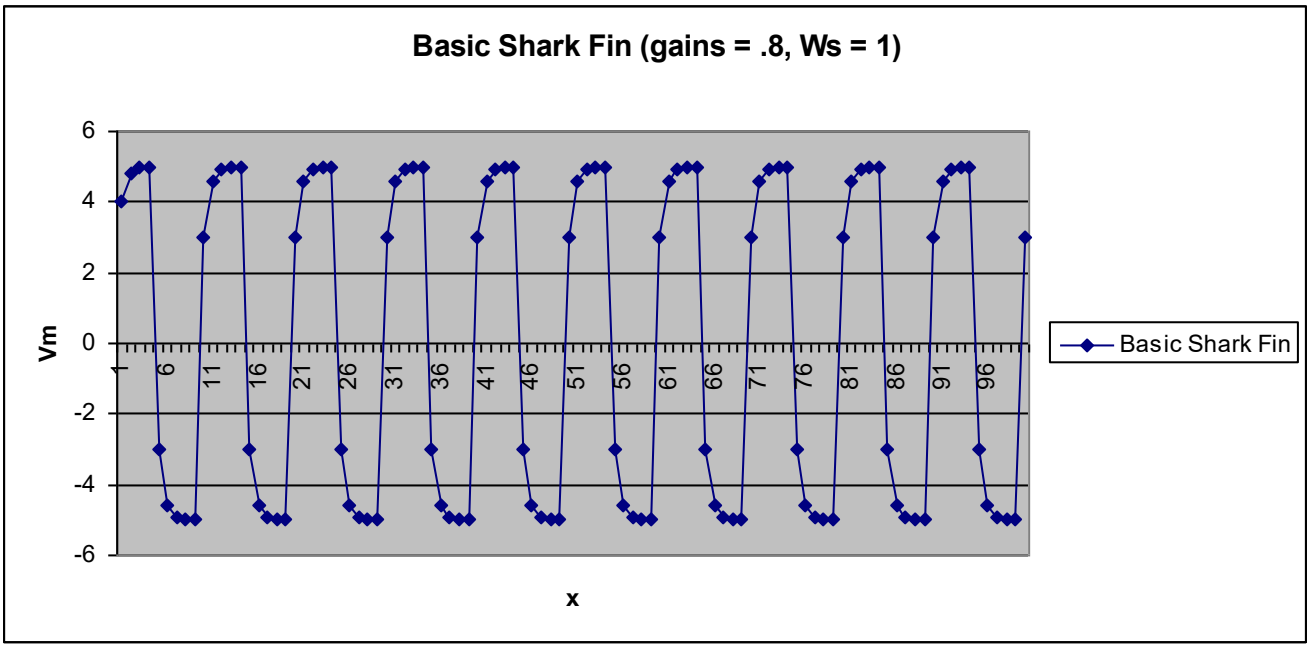

Figure 5. Output of the basic Shark Fin neural oscillator. Here the frequency of the waveform has been slightly more than doubled by doubling the gain parameter.

Observing the shape of the waveforms, it can be seen that the falling side of the waveform is inversely symmetrical to the rising side. This is due to the fact that both the Exciter and Inhibitor states use the same type of proportional controller strategy, and that the gains and inter-state weights are set equal to one another.

As we stated earlier, the approach that the Shark Fin oscillator uses is derived from the standpoint of a proportional closed loop control system. This means that the output value of the system, V_m, is incremented based on the difference between the current value of V_m and the requested value, which is V_{max} in the case of the Exciter state, and V_{min} in the case of the Inhibitor state. The system gain functions, gE and gI (gains for the Exciter state and Inhibitor state) control the rate at which V_m approaches V_{max} and V_{min}. The equations below, along with Figure 6, detail the computing processes of the Shark Fin oscillator. Figure 7 illustrates the effect of setting the inter-state weights to less than 1, it creates a waveform that dissipates over time. Figure 8 shows the effect of setting the inter-state weights to greater than 1, it creating an unstable growth of the waveform.

- V_{min} – Minimum value of waveform
- V_e – error
  - $V_e = V_{max} - V_m$ (Exciter state)
  - $V_e = V_{min} - V_m$ (Inhibitor state)
- g – gain
  - gE (gain Exciter state)
  - gI (gain Inhibitor state)
- V_c – Next requested value of waveform
  - $V_c = g*V_e$
    - Here the gain and error are specific to the current state, exciter or inhibitor.

- V_m – Measured value of waveform.
  - $V_m = V_m + V_c$
  - Here the assumption is that there is no load on the system. By this we mean that when we ask the system to increment the value of the waveform by V_c amount, we get the full amount added to the system. In a physical system, such as a truck going up a hill, when we request V_c more speed for the truck, we may or may not get the desired result, depending on the truck's load and mechanical characteristics.
- T – The tolerance value. Notice that the target values, V_{max} or V_{min}, can never be realized due to the nature of the equations below. V_m approaches V_{max} or V_{min} so in order to trigger the comparison that causes the oscillator to change states the value of V_m is compared to the target value less a tolerance value.
  - $V_m = V_m + (g*(V_d - V_m))$
    - V_d is the desired value, either V_{max} or V_{min}
    - g is the gain, either gE or gI

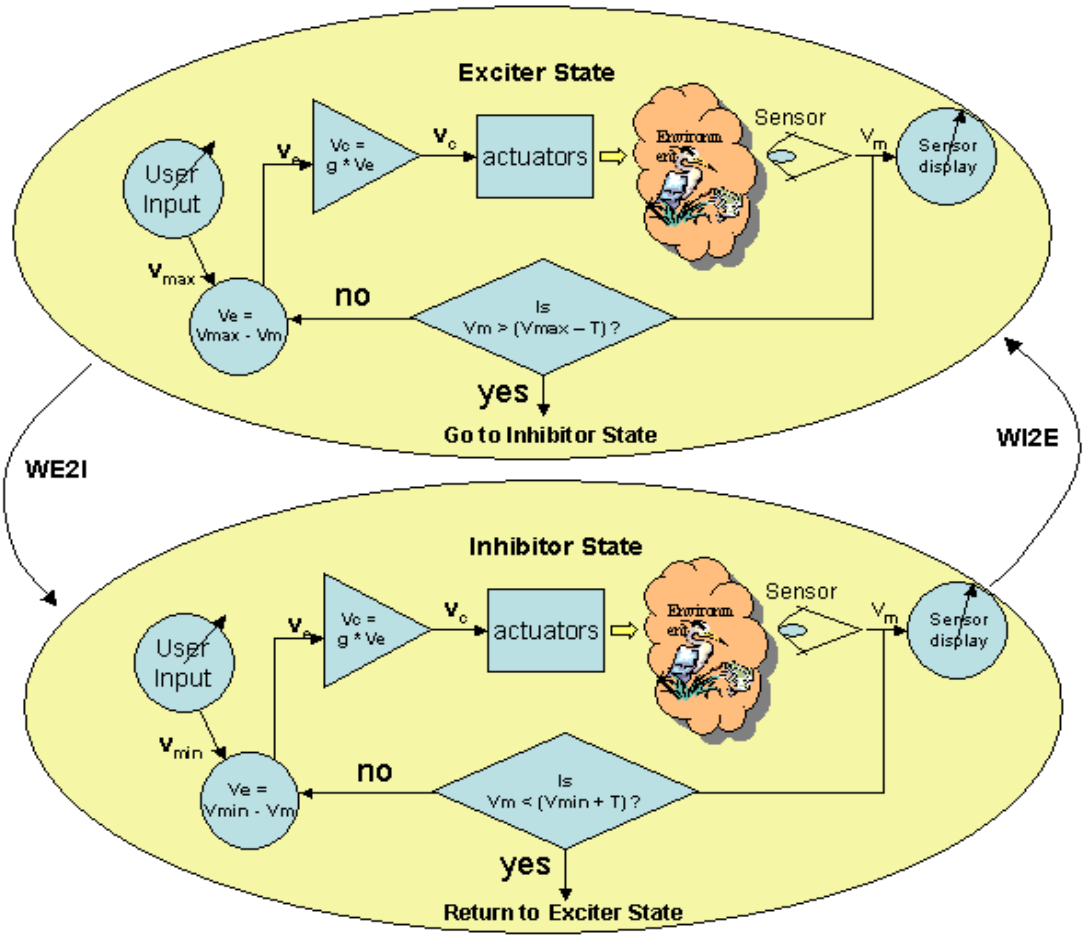

Figure 6. This figure shows an algorithm/data flow diagram for the Shark Fin neural oscillator. Note the proportional controllers used in each of the two states. Recall that when state transitions are made, the target values (either V_{max} or V_{min}) are multiplied by the weights that connect the states. That is V_{max} will be multiplied by W_{I2E} when control moves from the Inhibitor state to the Exciter state, and vice-versa. This is the mechanism in which a dissipating waveform is created.

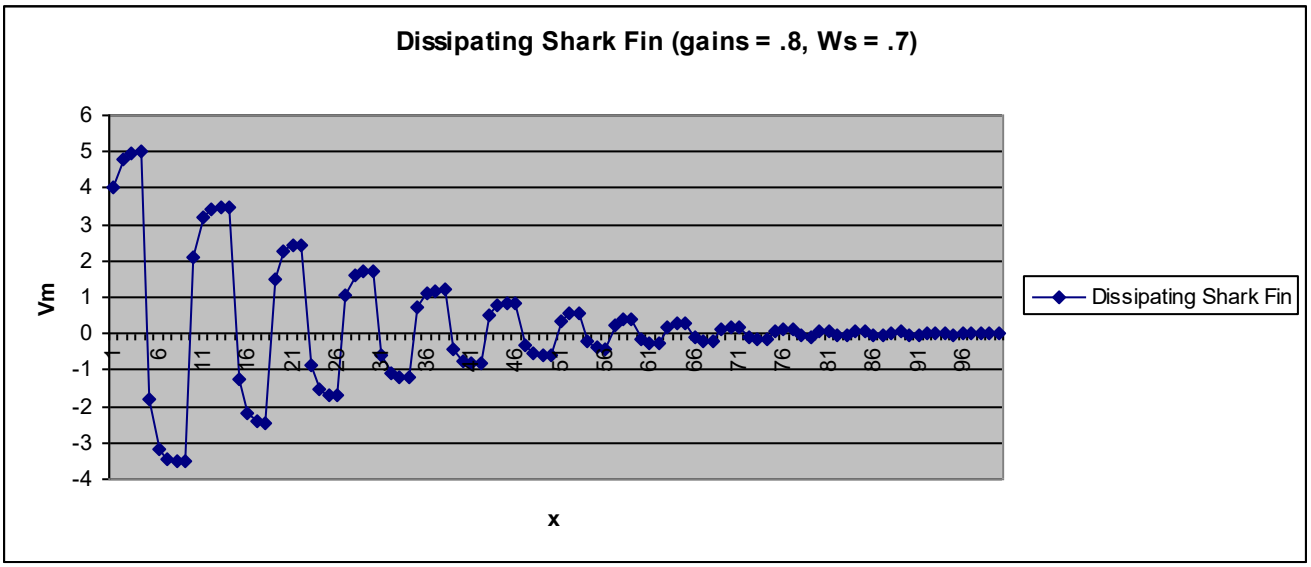

Figure 7. This figure illustrates the effect of setting the inter-state weights to less than 1; it creates a waveform that dissipates over time.

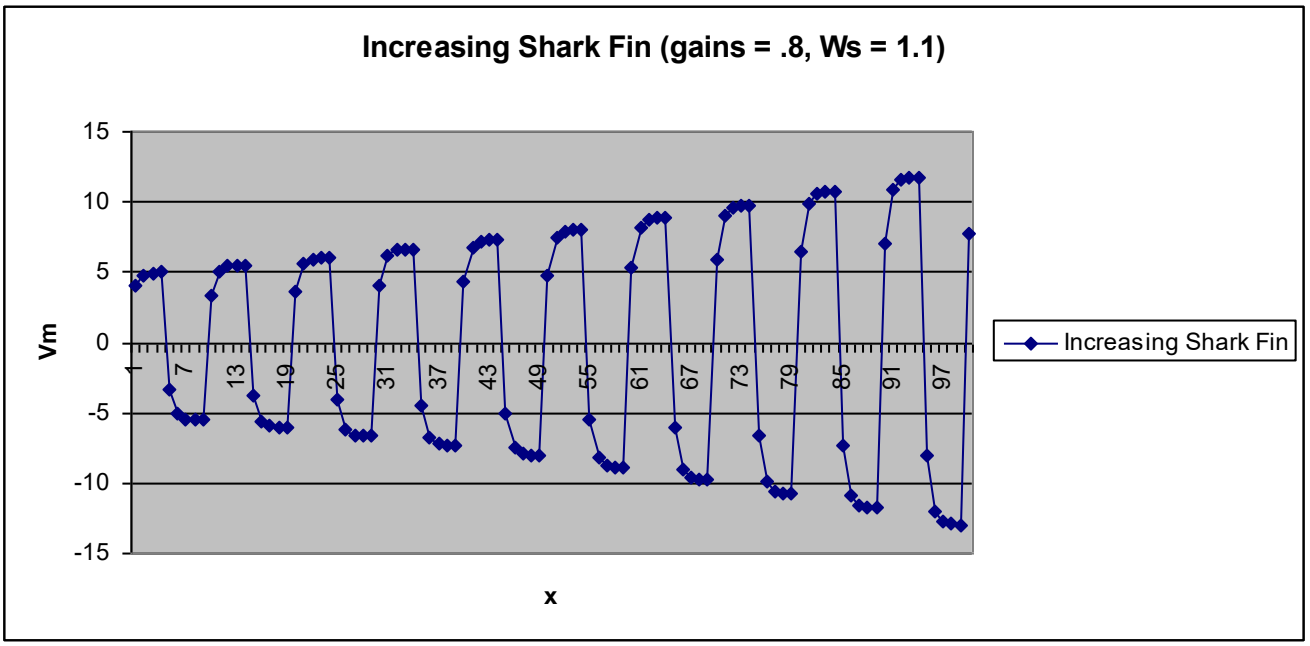

Figure 8. This figure shows the effect of setting the inter-state weights to greater than 1, creating unbounded growth in the waveform, thus an unstable condition.

3. Symmetric Waveform Oscillator with Four States

While the Shark Fin oscillator's waveform correlates well with the shape of several rhythmic biologic processes, including that of capnographic waveforms[16] (breathing patterns associated with bronchial spasms), many applications require a waveform with a smoother falling side. For example the oscillatory patterns required to control the segments of a swimming robot, or salamander such as that discussed by Ijspeert [5] are smooth on both the rising and falling sides of the waveform.

In order to produce a waveform of this nature, the transition from the Exciter phase to the Inhibitor phase has to be seamless. To extend the basic oscillator model to accommodate these requirements, from an intuitive standpoint two solutions come to mind:

- Immediately upon the transition from Exciter state to Inhibitor state, lower the gain.
- Select a new target value that is much closer to V_{max} than the original V_{min}. That is, work towards V_{min} using a series of "waypoints" in order to smooth the curve.

Shown in Figure 9 is the output of the "smooth4" oscillator, the enhanced version of the Shark Fin oscillator model. The key to making the transition smooth is the creating of a sub-state in each of the two primary states of the model, and the creation of a stable point of the waveform. The wording "stable point" in this context refers to a central location which can be used as a reference point during the migration of control between the states and sub-states of the oscillator model. To illustrate the operation of the system, consider the situation when V_m has just exceeded $V_{max} - T$. In this case, control is transitioning from the Exciter state to the Inhibitor state. Immediately upon entering the Inhibitor state, the Inhibitor's sub-state takes control. Instead of using the quantity V_{min} to calculate the error ($V_{min} - V_m$) the error is calculated by negating $V_{max} - V_m$. Remember, V_m is much closer to V_{max} at this point than it is to V_{min}, and this is what lets the system come down smoothly from its peak value. So, in essence, the second bullet from above is being used, but instead of looking forward to the target value, the system looks backward to the previous target value, and negates the result. As soon as V_m dips below the stable value (V_{stable}) the error is calculated in the same manner as that of the Shark Fin oscillator. Figure 10 illustrates the relationships between V_m, the stable point (V_{stable}), and the sub-states within the Excite and Inhibit states.

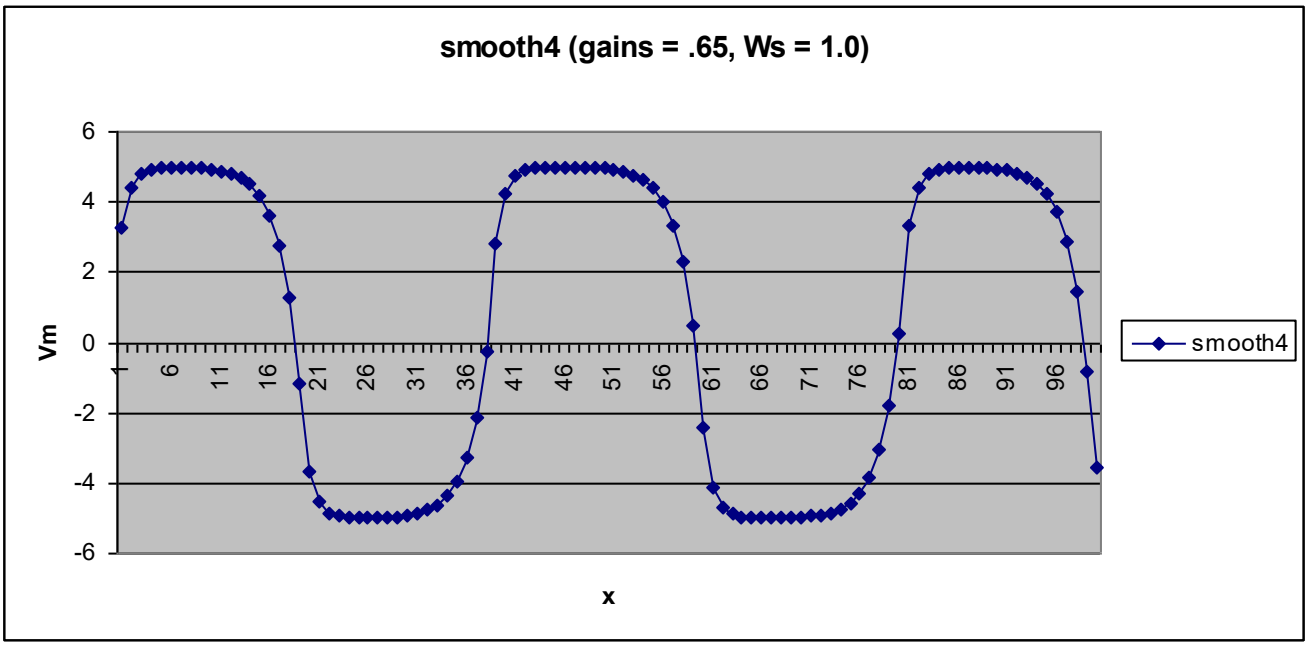

Figure 9. Output of the smooth4 neural oscillator. Here the waveform is continuous and non-diminishing. Notice the smooth transitions on both sides of the peaks and troughs of the waveform.

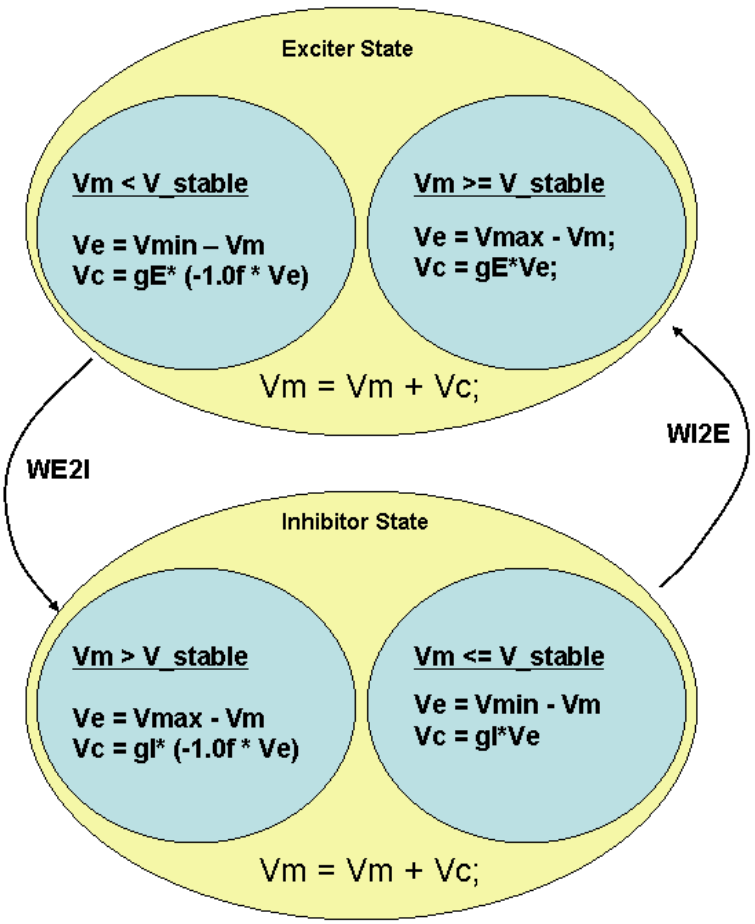

Figure 10. This figure shows an algorithm/data flow diagram for the smooth4 neural oscillator. As before, proportional controllers are used to generate values for V_m. However, this time two sub-states are contained in each of the primary states.

4. General Structure of Code for Basic Neural Oscillators

Below we show the general structure for the code for each of oscillators discussed in this article.

```
Inputs: Vmin, Vmax, inter-state weights, gains for each state
Outputs: Vm

state = excite
while (oscillating) {
        if (state is excite) {
                Vm = Calculate new value of system
                if (Based on Vm it is time to move to Inhibit state) {
                        modify Vmax based on Excite to Inhibit weights
                        set state to Inhibit
                }
        } /* End excite. */

        if (state is inhibit) {
                Vm = Calculate new value of system
                if (Based on Vm it is time to move to Excite state) {
                        modify Vmin based on Inhibit to Excite weights
```

```
                        set state to Excite
                }

        } /* End inhibit */

        determine if it is time to stop oscillating based on:
                number of iterations
                closeness of Vmax and Vmin to (Vmax + Vmin)/2
                Set oscillating variable appropriately.
} /* End while */
```

In order to begin the neural oscillator procedure outlined above, several input values must be determined. First of all it is necessary to set the target maximum and minimum output values, V_{max} and V_{min}, the weights between the states (normally these are equal), and the gains for each state (again, these are equal for many waveforms, but not all). In the literature, particularly when V_{max} and V_{min} are the same, this parameter is analogous to the "tonic" input. The output value V_m will be calculated at each iteration until it either reaches a fixed number of iterations, for a stable waveform, or when some stopping criteria is realized. For dissipating waveforms the output will reach a midpoint between V_{max} and V_{min}, at which point it will "flatline" and can be terminated. For increasing waveforms, a limit on the maximum amplitude of the output must be specified or an iteration limit given to end the oscillations before the waveform diverges too much.

The neural oscillator code begins by setting the current state to the Excite state. It will continually loop in this state, incrementing the output value V_m (in a way based on the desired waveform type) until it reaches its limit. At this point V_m is multiplied by the Excite-to-Inhibit weight and control is passed to the Inhibit state. It will continually loop in the Inhibit state, where output value V_m is decremented each iteration (in a way based on the desired waveform type) until its limit is reached. Then the output is multiplied by the Inhibit-to-Excite weight, and control returned again to the Excite state. The output at each iteration throughout the oscillation procedure can be output to a data file and/or fed into a graphical display procedure for real-time observation of the waveform and its behavior. The way that the output V_m is calculated within the states determines the overall shape of the waveform, and is the subject of the next section. The values specified for the gains and inter-neuron weights further shape the waveform in terms of its frequency and whether it will dissipate or increase in amplitude.

5.0 Waveform Tuning

As can be seen from figures 2, 4 and 9, the shape of the waveform can be highly dependent upon the implementation of the neural oscillator. In this section the effects of the various parameters on the shape, frequency and amplitude of the symmetric 4 state oscillator's waveform are illustrated. Curiously, this type of oscillator can produce surprisingly good approximations of the saw-tooth, shark-fin, and other waveforms, given the correct parameters. In Figure 11 below, the symmetric 4 state oscillator was used to produce a sine-like waveform (the line in blue), an approximation of a square wave (the pink waveform) and a waveform masquerading as saw-tooth or Shark Fin. The key

to changing the shape of the waveform lies in the manipulation of the gain parameters and T (the tolerance parameter).

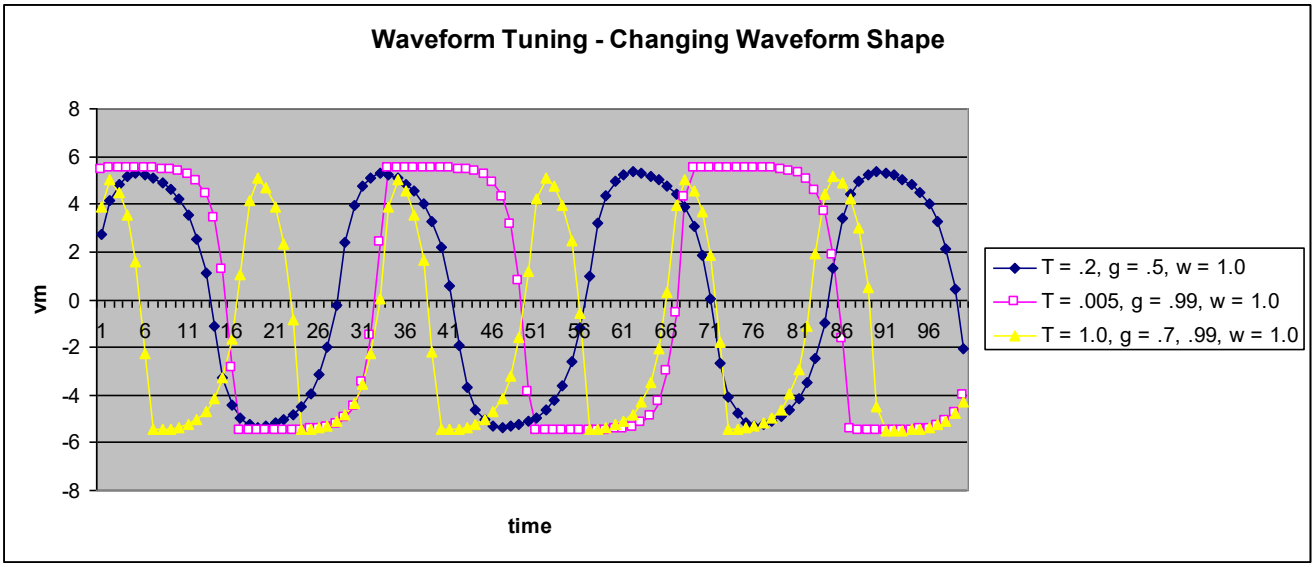

Figure 11. Various types of waveforms generated the symmetric 4 state oscillator. The different shapes are arrived at by manipulating the tolerance (T) and the gain (g). Of special note is the "saw-tooth like" waveform shown in yellow. It is the only one in which the gain for the rising side and falling side of the waveform are not equal.

The key to making the peaks and troughs of the waveform smooth, like those in a sine curve, lies in the T parameter. By making T large, the oscillator will transition states quickly, because the gain multiplied by the error can exceed the desired value (V_{max} or V_{min}) minus a large tolerance in fewer time steps than it can if T were small. Remember the current oscillator output, V_m, is not adjusted by a constant at each time step (as in the saw-tooth oscillator), but by a portion of the error, V_e (the difference between V_m and the target value). By setting T to a large value, the state transition is achieved while the error is relatively large. Conversely, by making T small, it will hold the oscillator very close to the transition point for an extended period of time, while V_m slowly edges up on $V_{max} - T$, or $V_{min} + T$. This can be seen in Figure 11. Notice how many more points there are across the top of the pink waveform than the blue or yellow waveforms. By combining a small tolerance with a large gain, the rising and falling sides are steep, but the crests and troughs are long and flat because it takes many time steps to whittle the error down enough to cause a state transition.

Changing the frequency is achieved by adjusting the gain parameter. Figure 12 below shows 3 "sine-like" curves of progressively higher frequency. All parameters for the curves are the same, except the gain.

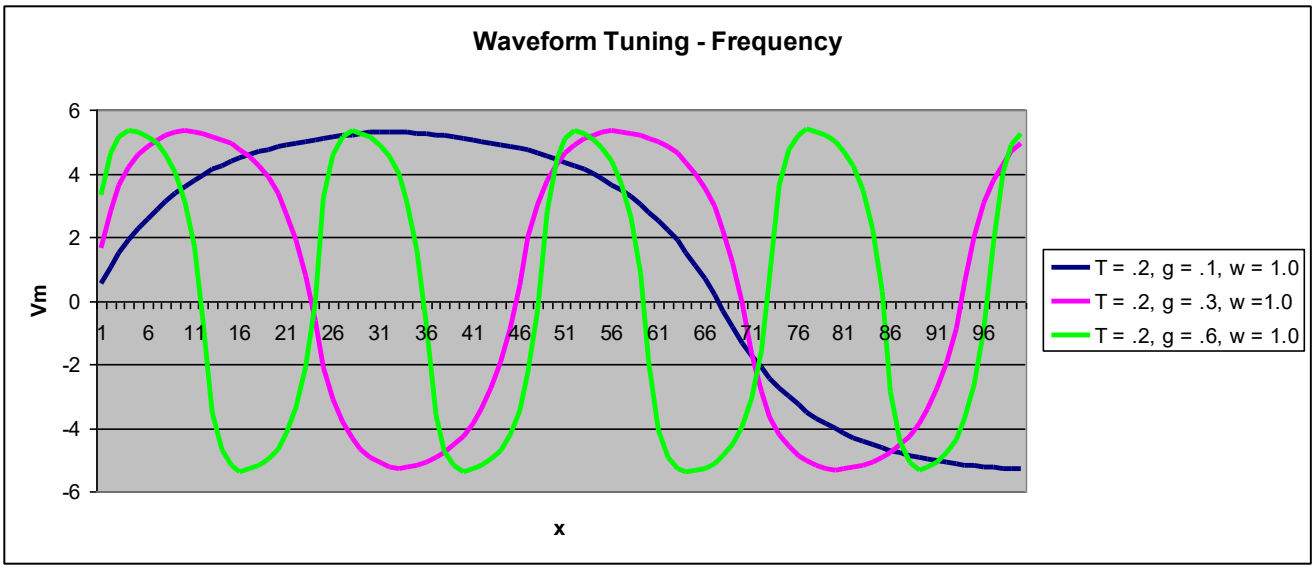

Figure 12. Frequency tuning using the gain parameter. Note that doubling the gain does not exactly double the frequency. This can be seen by counting crests of the pink and green curves.

Figure 13 illustrates how the amplitude can be manipulated by adjusting the desired value parameters, V_{max} and V_{min} (V_d in Figure 13). Both curves have all parameters the same, except their maximum and minimum values. This did cause a frequency shift, because it takes slightly less time to reach the state transition as the amplitude decreases. As earlier stated, in the literature this parameter is analogous to the "tonic" input.

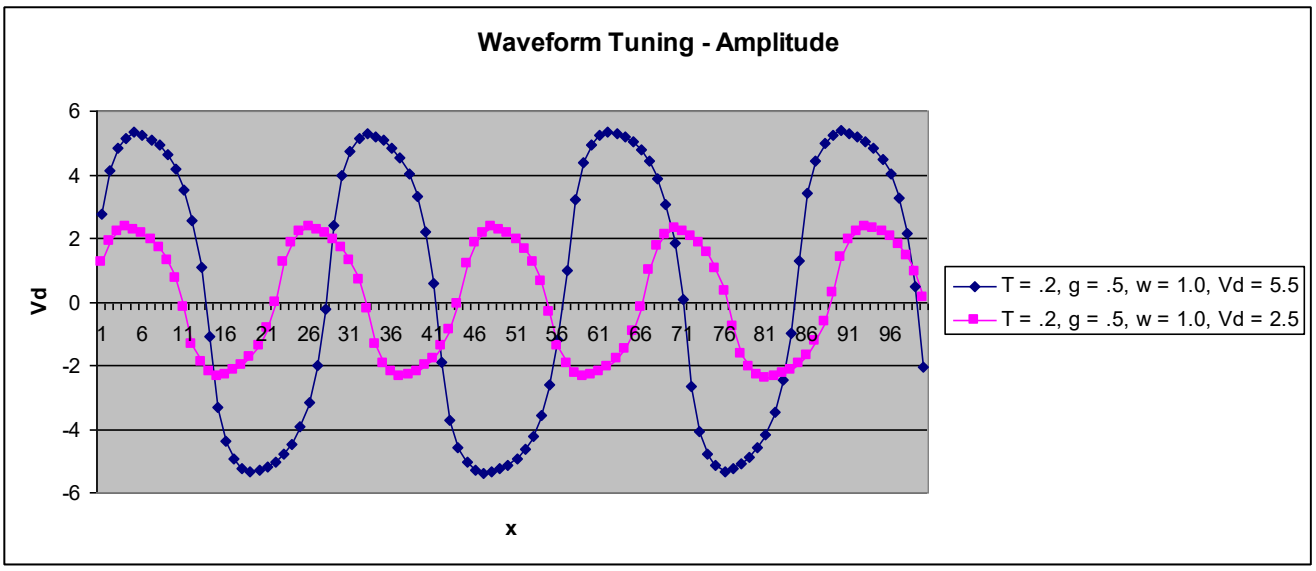

Figure 13. This figure shows the effect of changing the desired value (V_{max} and V_{min}) on the amplitude.

In summary, the waveform's shape, frequency, and amplitude, are easily adjusted using 3 types of parameters, the tolerance T, the gain g, and the desired value V_{max} and V_{min}. The code used to produce these pictures was set up so that each state had its own gain and desired value, but the tolerance is

shared. While the majority of the curves presented in this paper do not take advantage of this ability, it is still a valuable feature (the "saw-tooth like" of Figure 11 relies on it) for creating various shapes.

6. Conclusion

Neural oscillators are a fundamental component of robotics programming where robotic movement mimicking some form of natural rhythmic behavior is desired. There are numerous such models in the literature, varying in terms of the application, and even if their functionality is adequately explained, it is often the case that the actual programming of such models is not, often leaving the programmer at a loss as to where to begin.

The approach of this work was to detail the modeling of neural oscillators from a programmer's point of view. Here the basic components of the neural oscillator were presented and discussed in terms of the parts they play in the generation of the output signal. Variation of certain control variables, such as the inter-neuron weights or gains, or in the number of states, results in differently shaped waveforms. Here the basic saw-tooth waveform, a Shark Fin wave form, and a smooth curve waveform resulting as output from various neural oscillators were presented. Techniques for causing the waveform to dissipate in amplitude or to increasingly diverge were also discussed.

A generic procedure for programming a neural oscillator was presented, with information for varying the primary components to achieve the desired waveform shapes and strengths. Armed with this basic knowledge the robotics programmer can code simple to more complex neural oscillators for use in robotics applications.

Acknowledgement

This research was supported in part by Louisiana Board of Regents LEQSF (2007-10)-RD-A-27.

References

1. Arkin, R., Behavior Based Robots, MIT Press, Cambridge, MA, 1998
2. Cacciatore, T., Rozenshteyn, R., Kristan, W. Kinematics and modeling of leech crawling: evidence for an oscillatory behavior produced by propagating waves of excitation. *Journal of Neuroscience* **2000**, *20*, 1643-1655.
3. Chiel, J., Beer, R., Gallacher, J. Evolution and Analysis of Model CPGs for Walking: I. Dynamical Modules. *Journal of Computational Neuroscience* **1999**, *7*, 99-118.
4. Getting, P. Reconstruction of small neural networks. In *Methods in Neural Modeling;* Koch, C., Ed.; MIT Press, Cambridge, Mass., **1989**, 171-196.
5. Hoppensteadt, F.C., Izhikevich, E.M. Synaptic organizations and dynamical properties of weakly connected neural oscillators. *Biological Cybernetics* **1996**, *75*, 117-127.
6. Ijspeert, A.J.. A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biological Cybernetics* **2001**, *84*, 331-348.
7. Ijspeert, A., Kodjabachian, J. Evolution and Development of a Central Pattern Generator for the Swimming of a Lamprey. *Artificial Life* **1999**, *5,* 247-269.

8. Inada, H., Ishii, K. Behavior Generation of Bipedal Robot Using Central Pattern Generator (CPG). *Proceedings International Conference on Intelligent Robots and Systems* **2003**, Las Vegas, Nevada, 2179-2184.
9. Kleinfeld, D., Sompolinsky, H., Associative neural network model for the generation of temporal patterns: Theory and application to central pattern generators. *Biophys. J*. **1988**, *54*, 1039--1051.
10. Lockery, S., Sejnowski, T. The computational leech. *Trends in Neuroscience* **1993**, *16*, 283-290.
11. Meunier, D., Paugam-Moisy, H. Neural Networks for Computational Neuroscience. *European Symposium on Artificial Neural Networks – Advances in Computational Intelligence and Learning* **2008**, Belgium, 367-378.
12. Nakamura, Y., Mori, T., Masa-aki, S., Ishii, S. Reinforcement learning for a biped robot based on a CPG-actor-critic method. *Neural Networks* **2007**, *20*, 723-735.
13. Roberts, A., Tunstall, M. Mutual re-excitation with post-inhibitory rebound: A simulation study on the mechanisms for locomotor rhythm generation in the spinal cord of xenopus embryo. *European Journal of Neuroscience* **1990**, *2*, 11-23.
14. Selverston, A.I. Are central pattern generators understandable? *The Behavioral and Brain Sciences* **1980**, *3*, 535-571.
15. Wilson, D.M. The central nervous control of flight in a locust. *Journal of Experimental Biology* **1961**, *38*, 471-479.
16. You, B., Peslin, R., Duvivier, C., Vu, V., Grilliat, J., Medecine, H., Expiratory capnography in asthma: evaluation of various shape indices; *European Respiratory Journal*., **1994**, 7, 318-323.