Research Article

Neural Oscillators Programming Simplified

Patrick McDowell and Theresa Beaubouef

Department of Computer Science and Industrial Technology, Southeastern Louisiana University, Hammond, LA 70402, USA

Correspondence should be addressed to Theresa Beaubouef, tbeaubouef@selu.edu

Received 10 October 2011; Revised 29 December 2011; Accepted 2 January 2012

Academic Editor: Sebastian Ventura

Copyright © 2012 P. McDowell and T. Beaubouef. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The neurological mechanism used for generating rhythmic patterns for functions such as swallowing, walking, and chewing has been modeled computationally by the neural oscillator. It has been widely studied by biologists to model various aspects of organisms and by computer scientists and robotics engineers as a method for controlling and coordinating the gaits of walking robots. Although there has been significant study in this area, it is difficult to find basic guidelines for programming neural oscillators. In this paper, the authors approach neural oscillators from a programmer’s point of view, providing background and examples for developing neural oscillators to generate rhythmic patterns that can be used in biological modeling and robotics applications.

1. Introduction

Scientists have long employed neural oscillators as a method to study neuron/ganglia-based processes that serve as central pattern generators for various organisms and as a method to generate control and coordination signals for various robotic mechanisms. For example, significant work has been done in trying to understand the functions of various neurons and components of such biological neural networks [1, 2], developing properties general to all networks of neural oscillators [3], and the modeling of processes such as locomotion in simple animals [4–8]. Associative neural network models with behavior similar to central path generators have also been developed [9].

Computer scientists have researched and developed several varieties of artificial neural networks that model natural neural networks to some extent. Artificial neural network is a well-developed field, and the literature in this general area is quite vast; however, literature related to neural oscillators and central pattern generators using these computerized neural network models, such as in [10], is not so prevalent. Even so, the use of such models in controlling the motion of robots has been well established [11–13]. For researchers wanting to apply the techniques of neural oscillators and/or central path generators for the programming robotic controls, modeling rhythmic patterns, and so forth, without the necessity of understanding complex theoretical mathematical models or learning how simple invertebrates swim, there appears to be no single source of basic information available. Many articles provide basic diagrams of oscillators and occasionally some tuning parameters, but most do not provide discussions concerning the general nature of the mechanism, conceptual overviews, or implementation information. To help remedy this situation, this paper provides a basic nonbiological background on neural oscillators from a programmer’s perspective and discusses the design and coding of neural oscillators.

The paper is organized as follows. Section 2, Neural Oscillator Fundamentals, provides a basic introduction to the neural oscillator with an algorithm snippet for a simple sawtooth waveform. Building upon that, Section 3, Developing Symmetric Semicontinuous Waveforms, describes how to incorporate a proportional control strategy into the system so that the waveform is constructed with a series of continuous curves. Next, Section 4, Symmetric Waveform Oscillator with Four States, shows how to smooth the falling edge of the waveform so that continuous patterns can be generated. Section 5, General Structure of Code for Basic Neural Oscillators, provides the algorithm from which all neural oscillators described in the paper are based upon,
accompanying a brief descriptive narrative. Next, Section 6, Waveform Tuning, describes how to adjust the various parameters of the neural oscillator in order to shape the waveform. And, finally, Section 7, Conclusions, provides a brief summary of the paper.

2. Neural Oscillator Fundamentals

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 controlling 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 illustrates a typical two-state oscillator.

The general functioning of the oscillator is as follows. Starting the output value $V_m$ at an 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 illustrates the pattern with adding a constant $C = 1$ to the pattern. Notice that the pattern is not diminishing, that is, the amplitude remains constant, due to the interstate 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. (for the above example, see Algorithm 1).

3. Developing Symmetric Semicontinuous Waveforms

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, semicontinuous waveforms that can be generated is the “Shark Fin” waveform, as can be seen in Figure 4. This waveform is created by using a proportional controller [2] 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 5 illustrates the effects of increasing the gain of the controller. By increasing the gain, the frequency of the waveform is increased. Using the gain to tune, the waveform is more thoroughly discussed in Section 6.

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 Inhibiter states use the same type of proportional controller strategy and that the gains and interstate 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 $V_{\text{max}}$ in the case of the Exciter state and $V_{\text{min}}$ in the case of the Inhibitor state. The system gain functions, $gE$ and $gI$, are defined as:

$$
\begin{align*}
&gE = \frac{V_{\text{stable}} - V_{\text{min}}}{V_{\text{max}} - V_{\text{min}}} \\
&gI = \frac{V_{\text{max}} - V_{\text{stable}}}{V_{\text{max}} - V_{\text{min}}}
\end{align*}
$$

where $V_{\text{stable}}$ is the midpoint between $V_{\text{max}}$ and $V_{\text{min}}$. The system is initialized with the following variables:

- $V_{\text{stable}}$ = $(V_{\text{max}} + V_{\text{min}})/2.0f$ (set as the midpoint between $V_{\text{max}}$ and $V_{\text{min}}$)
- $V_m$ = $V_{\text{stable}}$ (output of the system)
- $V_{\text{d, work}}$ = $V_{\text{max}}$ (current max value for $V_m$)
- $V_{\text{0, work}}$ = $V_{\text{min}}$ (current min min value for $V_m$)
- state = “e” (state “e” is for exciter, i for inhibitor)
- $T$ = tolerance (set $T$ to the tolerance value, we want to reach $V_{\text{d, work}}$ and $V_{\text{0, work}}$ within tolerance value $T$)
- $C$ = increment value (set $C$ to the value that $V_m$ is incremented/decremented by)

The algorithm for generating patterns is as follows:

```c
/* Initialize variables. */
int j = 0; // j is the time variable (x-axis)
float V_stable = (V_{\text{max}} + V_{\text{min}})/2.0f; // V_stable is set as the midpoint between V_{\text{max}} and V_{\text{min}}
float V_p = V_{\text{stable}}; // V_p is the output of the system
float V_{\text{d, work}} = V_{\text{max}}; // the current max value for V_m
float V_{\text{0, work}} = V_{\text{min}}; // the current min min value for V_m
char state = "e"; // state “e” is for exciter, i for inhibitor
float T = tolerance; // Set T to the tolerance value, we want to reach V_{\text{d, work}} and V_{\text{0, work}} within tolerance value T.
float C = increment value; // Set C to the value that V_m is incremented/decremented by

/* Generate patterns. */
while (running) {
  /* Excite state. */
  if (state == "e") {
    increment V_m by amount C;
    /* If V_m exceeds the current max value for V_m, V_{\text{d, work}},
    transition to the inhibit state. */
    if ($V_m >= (V_{\text{d, work}} - T)$) {
      /* Set the state to inhibit. */
      state = "i"
      /* Diminish the current min value for V_m, V_{\text{0, work}},
      by the weight from Exciter to Inhibitor, W_{E2I}. */
      V_{\text{0, work}} = V_{\text{0, work}} * w_{E2I}
    }
  }
  /* Inhibit state. */
  else if (state == "i") {
    decrement V_m by amount C;
    /* If V_m is less than the current min value for V_m, V_{\text{0, work}},
    transition to the excite state. */
    if ($V_m <= (V_{\text{0, work}} + T)$) {
      /* Set the state to excite. */
      state = "e"
      /* Increase the current max value for V_m, V_{\text{d, work}},
      by the weight from Inhibitor to Excitor, w_{I2E}. */
      V_{\text{d, work}} = V_{\text{d, work}} * w_{I2E}
    }
  }
  /* Output data. */
  print(j, V_m)
  /* Increment count. */
  j = j + 1
}
```

Algorithm 1
waveform that dissipates over time. Figure 8 shows the effect of setting the interstate weights to less than 1; it creates an unstable growth of the waveform. The equations below, along with Figure 6, detail the computing processes of the Shark Fin oscillator. Figure 7 illustrates the effect of setting the interstate weights to greater than 1, creating a waveform is continuous and non-diminishing.

\[ V_{m} - 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81, 86, 91, 96 \]

\[-6, -4, -2, 0, 2, 4, 6, +6, +4, +2, 0, -2, -4, -6, \]

Figure 4: 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.

\[ V_{m} - 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81, 86, 91, 96 \]

\[-6, -4, -2, 0, 2, 4, 6, +6, +4, +2, 0, -2, -4, -6, \]

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.

gI (gains for the Exciter state and Inhibitor state) control the rate at which \( V_{m} \) approaches \( V_{\text{max}} \) and \( V_{\text{min}} \). The equations below, along with Figure 6, detail the computing processes of the Shark Fin oscillator. Figure 7 illustrates the effect of setting the interstate weights to less than 1; it creates a waveform that dissipates over time. Figure 8 shows the effect of setting the interstate weights to greater than 1, creating an unstable growth of the waveform.

(i) \( V_{\text{min}} \)—minimum value of waveform.

(ii) \( V_{e} \)—error.

(a) \( V_{e} = V_{\text{max}} - V_{m} \) (Exciter state).

(b) \( V_{e} = V_{\text{min}} - V_{m} \) (Inhibitor state).

(iii) \( g \)—gain.

(a) \( gE \) (gain Exciter state).

(b) \( gI \) (gain Inhibitor state).

(iv) \( V_{c} \)—next requested value of waveform.

(a) \( V_{c} = g \times V_{e} \).

(1) Here, the gain and error are specific to the current state, Exciter, or Inhibitor.

(v) \( V_{m} \)—measured value of waveform.

(a) \( V_{m} = V_{m} + V_{e} \).

While the Shark Fin oscillator’s waveform correlates well with the shape of several rhythmic biologic processes, including that of capnographic waveforms [14] (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 Hoppensteadt and Izhikevich [3] 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:

(i) immediately upon the transition from Exciter state to Inhibitor state, lower the gain,

(ii) select a new target value that is much closer to \( V_{\text{max}} \) than the original \( V_{\text{min}} \). That is, work towards \( V_{\text{min}} \) 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 creation of a substate 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 substates of the oscillator model. To illustrate the operation of the system, consider the situation when \( V_{m} \) has just exceeded \( V_{\text{max}} - T \). In this case, control is transitioning from the Exciter state to

(b) 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 \( V_{e} \) 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 \( V_{e} \) more speed for the truck, we may or may not get the desired result, depending on the truck’s load and mechanical characteristics.

(vi) \( T \)—the tolerance value. Notice that the target values, \( V_{\text{max}} \) or \( V_{\text{min}} \), can never be realized due to the nature of the equations below. \( V_{m} \) approaches \( V_{\text{max}} \) or \( V_{\text{min}} \) 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. For this application, \( T \) is set to 0.01.

(a) \( V_{m} = V_{m} + (g \times (V_{d} - V_{m})) \).

(1) \( V_{d} \) is the desired value, either \( V_{\text{max}} \) or \( V_{\text{min}} \).

(2) \( g \) is the gain, either \( gE \) or \( gI \).

4. 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 [14] (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 Hoppensteadt and Izhikevich [3] 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:

(i) immediately upon the transition from Exciter state to Inhibitor state, lower the gain,

(ii) select a new target value that is much closer to \( V_{\text{max}} \) than the original \( V_{\text{min}} \). That is, work towards \( V_{\text{min}} \) 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 creation of a substate 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 substates of the oscillator model. To illustrate the operation of the system, consider the situation when \( V_{m} \) has just exceeded \( V_{\text{max}} - T \). In this case, control is transitioning from the Exciter state to
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 $V_{\text{max}}$ or $V_{\text{min}}$) are multiplied by the weights that connect the states. That is, the current $V_{\text{max}}$ will be multiplied by $W_{12E}$ when control moves from the Inhibitor state to the Exciter state and vice versa. This is the mechanism in which a dissipating or increasing waveform is created.

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

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

the Inhibitor state. Immediately upon entering the Inhibitor state, the Inhibitor’s substate takes control. Instead of using the quantity $V_{\text{min}}$ to calculate the error ($V_{\text{min}} - V_m$), the error is calculated by negating $V_{\text{max}} - V_m$. Remember, $V_m$ is much closer to $V_{\text{max}}$ at this point than it is to $V_{\text{min}}$, and this is what lets the system come down smoothly from its peak value.
are used to generate values for the smooth4 neural oscillator. As before, proportional controllers transitions on both sides of the peaks and troughs of the waveform. The waveform is continuous and nondiminishing. Notice the smooth result. As soon as but instead of looking forward to the target value, the system 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 (\( V_{\text{stable}} \)), 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 (\( V_{\text{stable}} \)), and the substates within the Excite and Inhibit states.

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 (\( V_{\text{stable}} \)), 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 (\( V_{\text{stable}} \)), and the substates within the Excite and Inhibit states.

5. General Structure of Code for Basic Neural Oscillators

In the following we show the general structure for the algorithm for each of oscillators discussed in this paper.

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 (\( V_{\text{max}} \) and \( V_{\text{min}} \)), 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). 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 are realized. For dissipating waveforms, the output will reach a midpoint between \( V_{\text{max}} \) and \( V_{\text{min}} \), 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 is 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 interneuron weights further shape the waveform in terms of its frequency and determine whether it will dissipate or increase in amplitude.

6. 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, 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).

The key to making the peaks and troughs of the waveform smooth, like those in a sine curve, lies in the \( T \) parameter.
Inputs: $V_{\text{min}}, V_{\text{max}},$ inter-state weights, gains for each state
Outputs: $V_m$
state = excite
while (oscillating) {
  if (state is excite) {
    $V_m =$ Calculate new value of system
    if (Based on $V_m$ it is time to move to Inhibit state) {
      modify $V_{\text{max}}$ based on Excite to Inhibit weights
      set state to Inhibit
    }
  }/* End excite. */
  if (state is inhibit) {
    $V_m =$ Calculate new value of system
    if (Based on $V_m$ it is time to move to Excite state) {
      modify $V_{\text{min}}$ 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 $V_{\text{max}}$ and $V_{\text{min}}$ to $(V_{\text{max}} + V_{\text{min}})/2$
  Set oscillating variable appropriately.
}/* End while */

\section*{Algorithm 2}

\begin{figure}[h]
  \centering
  \includegraphics[width=\textwidth]{figure11.png}
  \caption{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.}
\end{figure}

By making $T$ large, the oscillator will transition states quickly, because the gain multiplied by the error can exceed the desired value ($V_{\text{max}}$ or $V_{\text{min}}$) 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, $V_{\epsilon}$ (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 $V_{\text{max}} - T$ or $V_{\text{min}} + T$. 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 they take 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 three "sine-like" curves of progressively higher frequency. All parameters for the curves are the same, except the gain.

\begin{figure}[h]
  \centering
  \includegraphics[width=\textwidth]{figure12.png}
  \caption{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.}
\end{figure}

Figure 13 illustrates how the amplitude can be manipulated by adjusting the desired value parameters, $V_{\text{max}}$ and $V_{\text{min}}$. \(T = 0.2, g = 0.5, w = 1\)
\(T = 0.005, g = 0.99, w = 1\)
\(T = 1, g = 0.7, 0.99, w = 1\)
The tolerance and amplitude are easily adjusted using three types of parameters: the tolerance $T$, the gain $g$, and the desired value $V_{\text{max}}$ and $V_{\text{min}}$. This did cause a frequency shift, because it takes slightly less time to reach the state transition as the amplitude decreases. In the literature, this parameter is analogous to the “tonic” input.

In summary, the waveform's shape, frequency, and amplitude are easily adjusted using three types of parameters: the tolerance $T$, the gain $g$, and the desired value $V_{\text{max}}$ and $V_{\text{min}}$. The values $V_{\text{max}}$ and $V_{\text{min}}$ are determined by the system that is being modeled. These two parameters, in conjunction with the $T$ parameter, will determine the amplitude of the waveform. The $T$ parameter also helps determine how smooth the waveform is by helping to generate more points at the peaks and valleys. Another point to note is that the tolerance value will be proportional to the range of the waveform. For example, if the tolerance value is 25% of the range of the waveform, the transition from excite state to inhibit will be much more abrupt than if the tolerance value is 1% of the range. That is, if the range of the waveform is $2a$, a tolerance value of $.5$ will cause a much abrupt transition than a tolerance value of $.1$ in a waveform whose range is 50. And, finally, the gain parameter directly affects the frequency of the system. 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 waveform” of Figure 11) for creating various shapes.

### 7. Conclusions

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.


Submit your manuscripts at http://www.hindawi.com