Chapter 7
Neural Oscillators

A brief Introduction

Neural oscillators are all around us, in fact some of the most important ones are *inside* us, I am thinking of the neural oscillators which make our heat beat, our lungs breathe, our gut transport food down the digestive tract, and of course the neural circuits which drive the motor neurons in our legs to make us walk. Neural oscillation also features in many disorders such as synchronization of brain neurons during epileptic seizures, tremors in Parkinson's disease patients, and disruption of cortical oscillation in schizophrenia.

All of the above are of great interest within computing, where computational solutions of the underlying mathematical models will further our understanding. But perhaps the most interesting application of modelling neural oscillators is in robotics, in particular those circuits which enable legged robots to walk, segmented robots to swim and crawl. This application area is called 'Central Pattern Generators' (CPGs). The idea is that a small neural circuit will drive tetrapod/quadruped legs (think a horse) to make it move with a number of different *gaits* (canter, trot, gallop). In this chapter we shall work up to an understanding of hexapod (beetle) gaits.

Oscillation Refresher

Before we get into the details, let's just review our understanding of simple *mechanical* oscillations with which we are all familiar. Let's take a car suspension as an example, Fig.1 shows a Land Rover spring and a simple model, the Land Rover body mass (yellow) on the spring, the bottom of which is resting on the ground (we've neglected the wheel and tyre in the model). This is a simple *mass-spring* system.

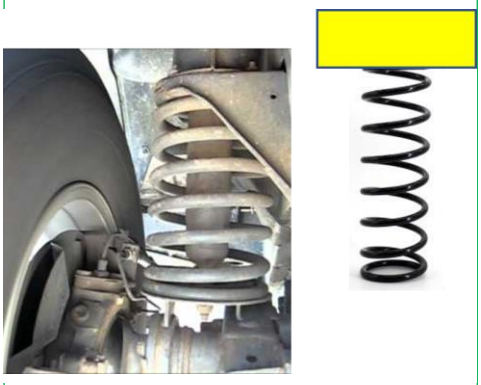
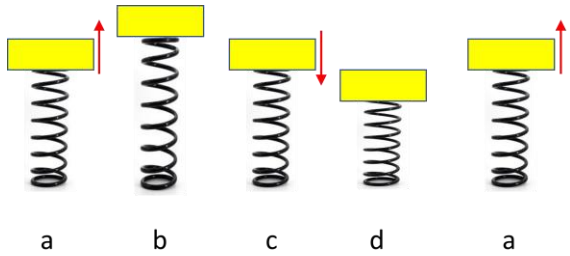


Figure 1 Land Rover Suspension

When you are driving along and hit a bump, then the spring gets compressed, and the car body will oscillate up and down as shown in the diagram below. The body starts at **a** and as time progresses, passes through **b, c, d,** and back to **a** where the cycle of oscillation starts again. Note the red arrows show how the body is moving at these snapshots in time.



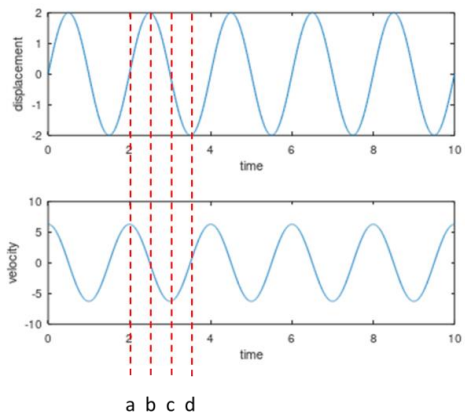


Figure 2 Displacement and Velocity of an oscillating Land Rover

When we model the suspension (mathematically) and plot out the solution, then we find the *displacement* and *velocity* of the car body change as shown in Fig.2. These have been labelled according to the above diagram. What do these tell us? This is not easy to answer, since we must look at two plots and the diagram, a bit like playing the organ[1].

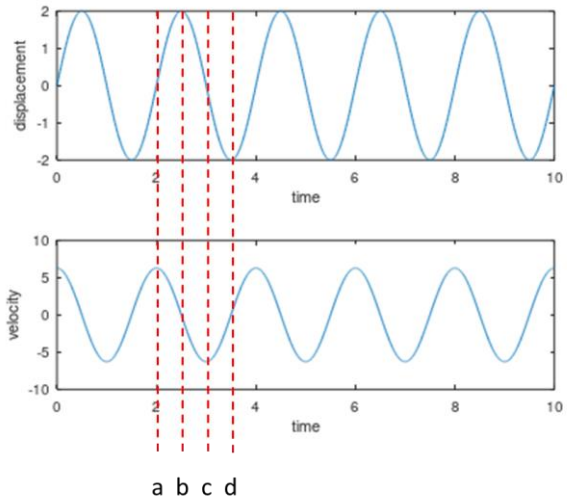
Well at **a** there is no displacement, but the Land Rover body has positive velocity which means it moves upwards until it reaches its maximum displacement at **b.** Here its velocity is zero. Then the body velocity becomes negative, it is moving down. So its displacement decreases, and it passes through zero displacement **c** where the velocity is its largest (negative) value. Then it slows down, and at **d** it has stopped (zero velocity) with the largest negative displacement. Then its velocity becomes positive, so it rises again until it hits its starting state **a**. One cycle of oscillation is complete, the rest is repetition!

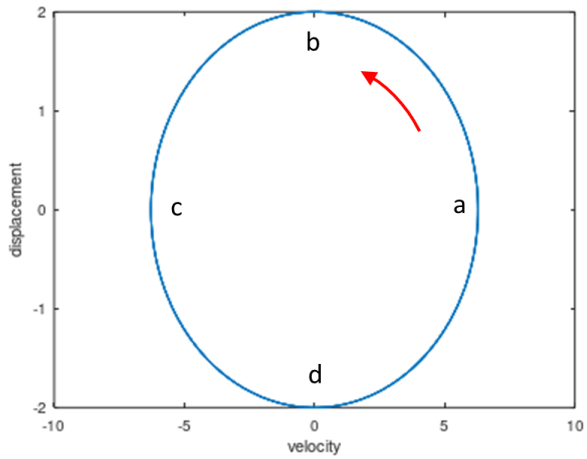
Looking at Fig.2 we see something interesting. Both displacement and velocity are oscillating with the same period, around 2 sec. The peak of the velocity seems to 'follow' the peak of the displacement. So, displacement and

[1] I am learning the organ at the moment, so you have my symphony.

velocity are both *periodic in time*. They are clearly related, so this raises the question how could we show this relationship? The answer is fundamentally important to understand oscillating systems, this is the *concept* of the ***phase plane***.

The diagram below reproduces Fig.2 with the addition of the *phase plane* on the right, a plot of displacement versus velocity.





This result is quite amazing; we get a closed curve! We start at **a** then progress through **b, c, d** and back to **a**. Then we go around the curve again (the red arrow shows the direction of motion around this curve). This works because both displacement and velocity are periodic, with the same period.

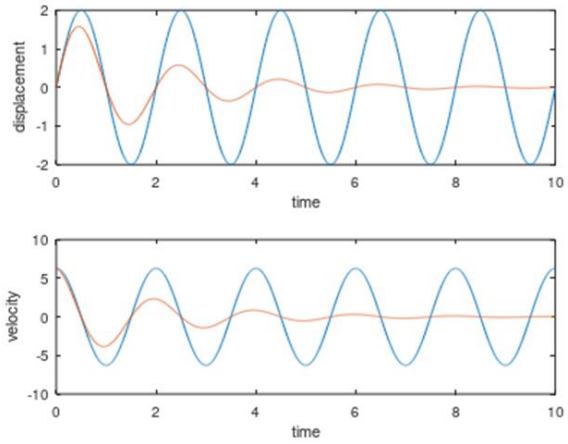
This *trajectory* of our suspension system in the *phase plane* is quite fundamental, and in general tells us a lot about an oscillating system. The phase plane combines displacement and velocity, but we have lost an explicit representation of time. Of course, we can recover this by using a coded solution.

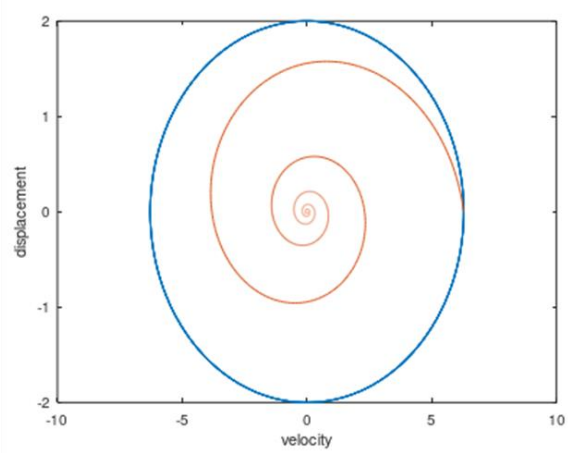
Oscillator_Phase_Plane.m

So, let's develop this further for our Land Rover Suspension system. When the Land Rover hits a bump, it starts to oscillate, but the oscillations are damped (removed) by the

suspension shock absorbers. Let's see how the above diagrams change.

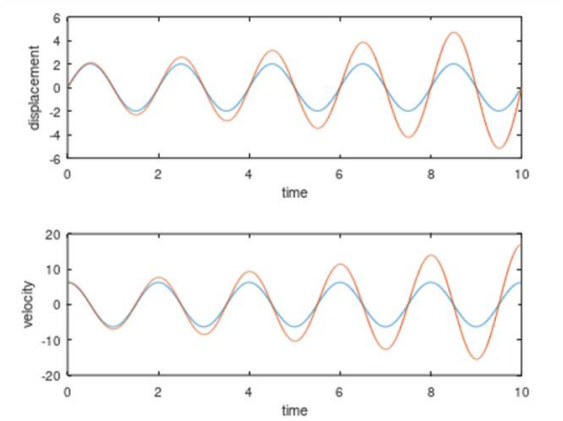
The blue plots reproduce the above, the red plots show the effect of damping. Both the Land Rover body displacement and velocity decay down to zero; it stops moving (zero velocity) and approaches its 'sleeping' location (zero).

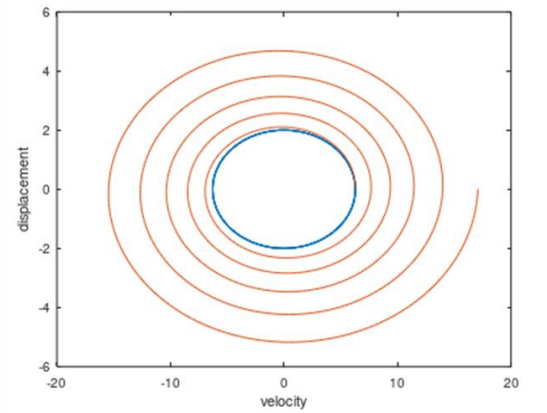




All this information is contained in the phase plot. The red spiral trajectory, moving inwards (0,0) clearly shows that both the Land Rover suspension displacement and body velocity decrease with time and end up at zero.

Now let's do something hypothetical and see what happens if we add energy into the suspension system rather than taking it away. A simple computation yields the following result.





Both displacement and velocity increase with time due to the added energy, the trajectory on the phase plane is a spiral expanding outwards. This is quite an unstable system and would spell disaster for a physical suspension system.

In summary, we see three sorts of trajectories on the phase plane. First is the stable closed ellipse. In general, this can be any smooth shape, and is called a ***limit cycle***. The other trajectories either move towards this limit cycle or move away from it.

Building an Oscillator from Leaky Integrators

Here we shall attempt to build a neural oscillator by connecting two leaky integrator neurons together. We shall discover that it is possible to create an oscillator, but also that this is useless, and cannot exist in a real biological system.

The first thing to note is that a single leaky integrator neuron cannot oscillate; we have investigated single neurons in detail, and we have never experienced oscillation. So, we turn to two neurons and think how to make these oscillate. If we put them in a chain, one after the other then again, we will not get oscillations. The only possibility is if we connect the output of the second to the input of the first, so we have a feedback loop, rather like a microphone close to a loudspeaker which produces a howl.

Let's take two neurons and consider the general case where each neuron is connected to each other neuron, including itself. This is shown in Fig.3. The strengths of the connexions between the neurons are shown as the 'w' symbols, where w_{ij} is the strength of connexion from neuron j into neuron i. So w_{12} concerns the output of neuron-2 coming into the input of neuron-1, and w_{11} shows the input of neuron-1 from its own output.

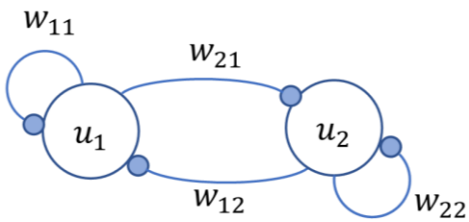


Figure 3 Two neurons totally connected with each other

Now let's write down some possible equations for two leaky integrators connected. This is straightforward and follows from the material presented in Chapter 5.

$$\frac{du_1}{dt} = \frac{1}{\tau}(-u_1 + w_{12}u_2)$$

$$\frac{du_2}{dt} = \frac{1}{\tau}(-u_2 + w_{21}u_1)$$

The first terms on the right in both equations are the integrator leak, and the second terms are the coupling from the other neuron. So, we view the leak as a neuron coupled to itself. We can write the above equations in matrix form

$$\begin{bmatrix}\dot{u}_1 \\ \dot{u}_2\end{bmatrix} = \begin{bmatrix}-1 & w_{12} \\ w_{21} & -1\end{bmatrix}\begin{bmatrix}u_1 \\ u_2\end{bmatrix}$$

Now, the theory of linear systems (see appendix) tells us that if this system will oscillate then the *trace* of the matrix must be zero, Here the trace is -1 + -1 which is -2 which is not zero. So, we must add a positive coupling from at least one neuron into itself, let's call this *a* and do this for neuron-1. Then we have

$$\begin{bmatrix}\dot{u}_1 \\ \dot{u}_2\end{bmatrix} = \begin{bmatrix}a-1 & w_{12} \\ w_{21} & -1\end{bmatrix}\begin{bmatrix}u_1 \\ u_2\end{bmatrix}$$

Now the trace is (a – 2) and since this has to be zero, we find that a = 2. No other value will do. This turns out to be a real problem, but we shall come back to this, since there is another condition on the above system if it is to oscillate. The *determinant* of the matrix must be greater than zero. For the above matrix this means

$$(a-1)(-1) - w_{12}w_{21} > 0$$

which simplifies to

$$w_{12}w_{21} < -1$$

This result tells us that one of the interconnections must be excitatory and one inhibitory, so the product is a negative number (and less than -1). So, our leaky integrator oscillator circuit looks like Fig.4 where excitatory and inhibitory synapses have been clearly indicated.

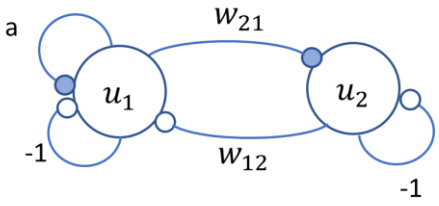


Figure 4 Leaky integrator oscillator circuit

While this neural circuit will work, it will only work in theory. The problem is that the value of *a* needs to be fixed to an *exact* value, $a = 2$. Mathematically this is possible, but it is infeasible in a biological system which functions using chemicals whose concentrations may have a range of values. Your heart does not stop beating when you have a cup of coffee (which affects levels of methylxanthine in your system). A similar conclusion could be reached thinking about solving the above system on a digital computer; you set the value of *a* like this `a = 2.0;` but the CPU represents this number with a finite number of bits, and there is no guarantee that its representation is exactly 2.0. Glance at Fig.5 which shows my computer's solution which is not a steady oscillation. The behaviour is hardware dependent, it is not robust to changes unlike a biological system. My heart beats, so does yours and at about the same rate.

There is another *nasty* property of the leaky-integrator oscillator, which is also shared by the Land Rover suspension. This concerns the *amplitude* of the oscillation. The above theory does not predict a value for the amplitude, so we conclude this could be anything from 0.1 to a million. The trajectory in phase space could be of any size. Biology does not like such unknowns; all our hearts beat with approximately the same amplitude.

Biological Neural Oscillators

Recordings of neural oscillators *in vivo* have been made at a number of scales from small collections of neurons to larger interacting populations, especially in the brain cortex. These have been successfully modelled such as the FitzHugh – Nagumo model of small collections, or the Wilson-Cowan equations for the cortex. Studies reveal that real neurons emit periodic pulses or bursts of oscillations. These are called 'spiking' neurons, see Fig.5. Computational models of spiking neurons are used mainly to understand biological neurons rather than applications in engineering scenarios such as Central Pattern Generators for legged or segmented

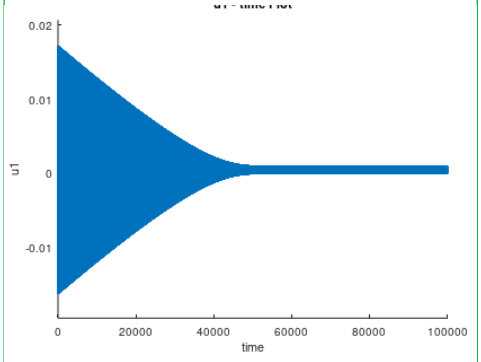


Figure 6 Solution of leaky integrator oscillator where the value of a has been set to 2.0 in the code.

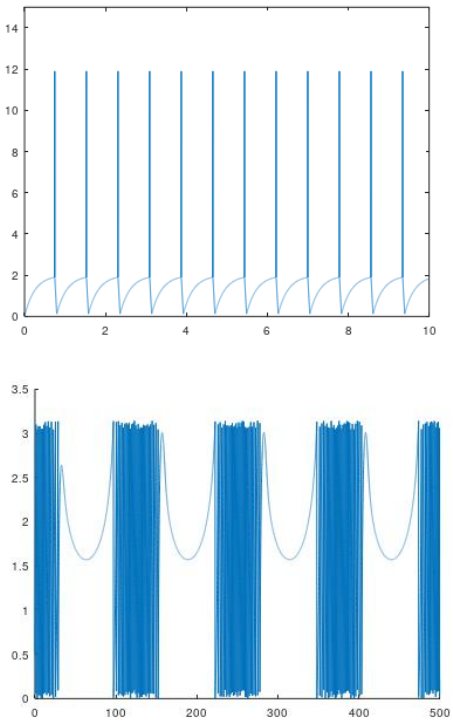


Figure 5 Simulation of 'spiking' neurons. Top Integrate and Fire, bottom Atoll models

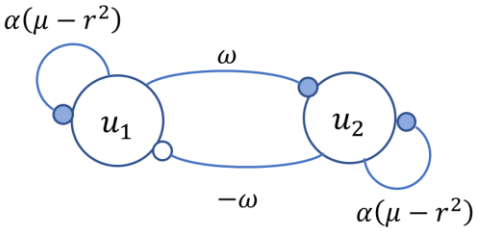


Figure 7 Structure of the Hopf neural oscillator

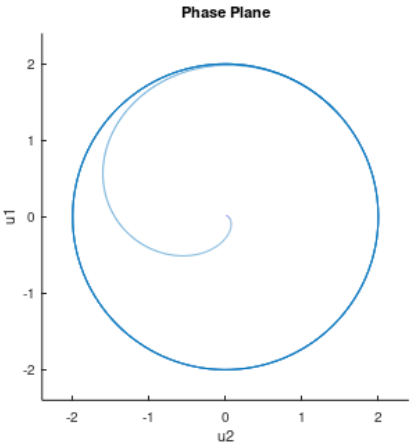


Figure 9 Limit Cycle for the Hopf oscillator

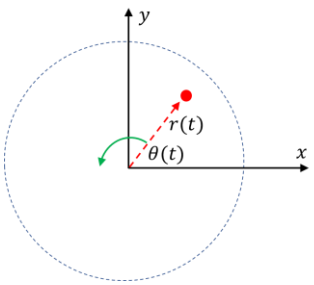


Figure 8 Polar coordinate description of the phase plane

robot motion. We shall study the simplest model which captures the fundamental behaviour of most neural oscillators, the Hopf oscillator. Later we shall apply this to the locomotion of a hexapod robot.

The Hopf Oscillator

This comprises a pair of neurons, and its structure is very similar to the leaky-integrator model discussed (and rejected) above. Each neuron influences itself and is connected to the other neuron. Like the leaky-integrator oscillator, one of these inter-connections is excitatory and the other is inhibitory. Here are the equations for the neurons

$$\frac{du_1}{dt} = \alpha(\mu - r^2)u_1 - \omega u_2$$

$$\frac{du_2}{dt} = \omega u_1 + \alpha(\mu - r^2)u_2$$

where there are three parameters, α, μ and ω and the variable r is just

$$r = \sqrt{u_1^2 + u_2^2}$$

Looking at the two ODEs, the feedback term in each neuron depends on $(\mu - r^2)$ so if r is small then this is positive and the values of both neurons will grow. But if $r^2 > \mu$ then the feedback is negative, and the values of both neurons will fall. In fact the trajectory in the phase plane moves to a circle of radius $\sqrt{\mu}$. This is a stable *limit cycle* which attracts all trajectories in the phase plane, Fig.8. The above ODEs for the neurons can be transformed to polar coordinates (radius r angle θ) as shown in Fig.9.

$$\frac{dr}{dt} = \alpha(\mu - r^2)r$$

$$\frac{d\theta}{dt} = \omega$$

The first expression shows how the radius of the trajectory point changes with time, and the second expression shows how the angle changes with time. You can see that the angle changes at a constant rate ω which defines the frequency of the oscillator. The right-hand-side of the first expression is positive if $r^2 < \mu$ causing r to increase, and it's negative if $r^2 > \mu$ which causes r to decrease. When $r^2 = \mu$ then the right-hand-side is zero, so r does not change. This shows that the trajectory will always approach the limit cycle, which for the Hopf oscillator is a circle.

Phase Lag Neurons

When we come to modelling insect gaits, we shall find that their legs oscillate with the same frequency, but there is a phase lag between legs, e.g., a 180 degree phase lag means one leg is moving forwards (raised off the ground) and the next one is moving backwards (on the ground) so the bug moves forwards.

This can be achieved with a single leaky-integrator neuron

$$\frac{du}{dt} = \frac{1}{\tau}(-u + I)$$

where the input is e.g., $I = \sin \omega t$ which is a sine wave with frequency ω. The output u of the neuron is another sine wave with amplitude

$$\frac{I}{\sqrt{1 + \tau^2 \omega^2}}$$

which is always less than I and with a phase lag given by

$$\tan \varphi = \omega \tau$$

Now we use these results to *engineer* a neuron which provides us with a desired phase lag but keeps the amplitude constant. Since the frequency of the sine wave is given (e.g., by the Hopf oscillator) then we use the time constant τ to set our desired phase,

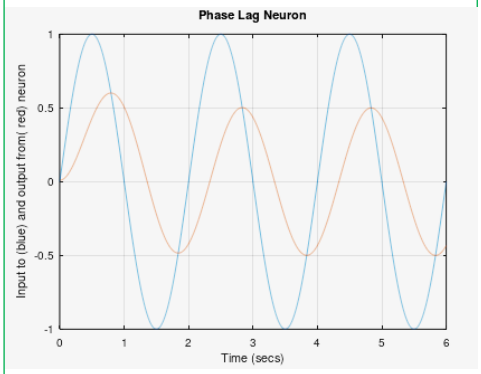


Figure 10 Input (blue) and output (red) from a single neuron giving a phase difference of 60 degrees

$$\tau = \frac{\tan \varphi}{\omega}$$

Now to restore the amplitude to its original value *I* we must multiply the input to the neuron by the factor by which the amplitude was decreased, namely $\sqrt{1 + \tau^2 \omega^2}$. So, our engineered neuron becomes

$$\frac{du}{dt} = \frac{1}{\tau}(-u + kI)$$

where

$$k = \sqrt{1 + \tau^2 \omega^2}.$$

This neural circuit works well, it was used to create the phase lag shown in Fig.10.

There is one complication; due to the nature of the tangent function, this circuit can produce a maximum phase shift of 90 degrees, and often we need more than that. The solution is to use a chain of, say 3 neurons, and get each to provide 1/3 of the total lag we need. The neural circuit sketched below shows just such a chain with the sine wave produced by our friend, the Hopf oscillator.

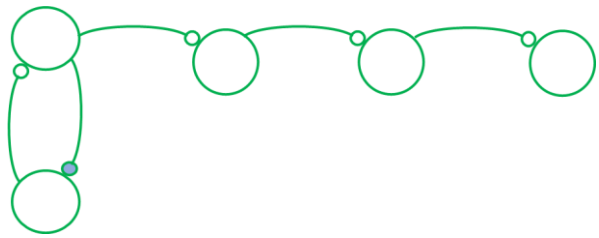


Fig.11 shows the output of such a chain which was asked to produce a total phase shift of 180 degrees. You can see this by comparing the light blue curve (input) and the purple curve (output). The only slight issue is that it takes a certain time for the chain to settle, there is a clear initial transient which lasts around 4 seconds or so.

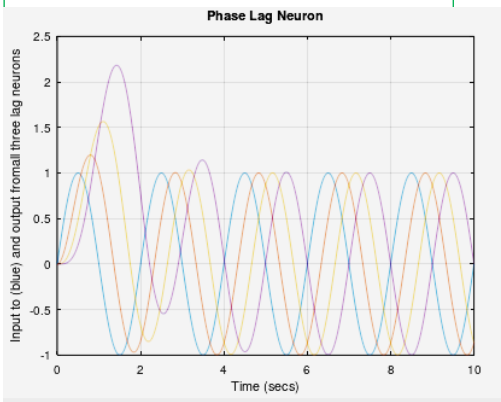


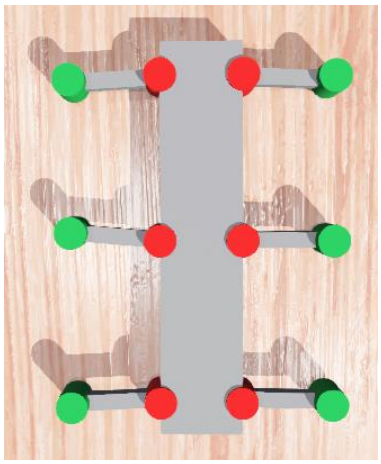
Figure 11Phases along chain of three neurons, the total phase lag is 180 degrees

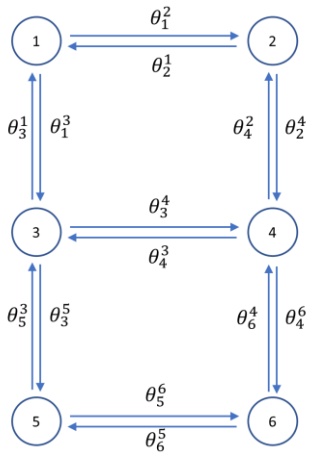
Insect Gait Patterns – The Hexapod

Previous Work

Biological systems (creatures) have provided much inspiration for generations of engineered robots that can walk, crawl or swim. One of the most influential observations for segmented creatures such as fish was the propagation of a wave of oscillation down the length of the creature. If the creature were cut into two, then each part would continue to show this wave. These observations led to the idea that the neural circuit consisted of a *coupled chain of oscillators* rather than just one master oscillator.

Most previous work has adopted this architecture, e.g., the hexapod shown below comprises six oscillators (neuron pairs) one for each leg.





The neural circuit for this model is quite complex, the equation for each neural oscillator has the form

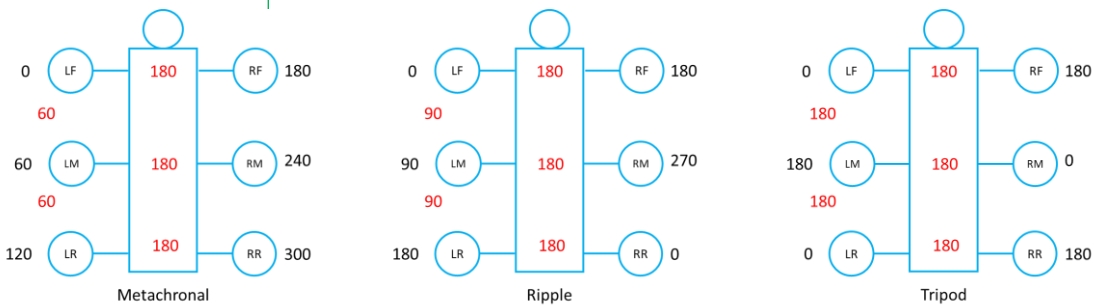
$$\begin{bmatrix}\dot{x}_i\\ \dot{z}_i\end{bmatrix} = \begin{bmatrix}\alpha(\mu - r_i^2) & -\omega\\ \omega & \alpha(\mu - r_i^2)\end{bmatrix}\begin{bmatrix}x_i\\ z_i\end{bmatrix} + \sum_{j\neq i} \boldsymbol{R}\left(\theta_j^i\right)\begin{bmatrix}0\\ \dfrac{x_j + z_j}{r_j}\end{bmatrix}$$

where the rightmost term captures the coupling into the *i*'th oscillator from the other oscillators connected to it. You will recognize the Hopf oscillator here. See appendix for maths.

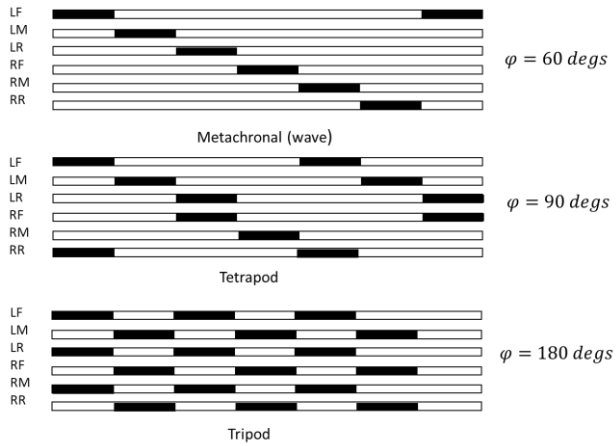
Our Approach

We do not need a chain of coupled oscillators since we shall agree not to cut up our robot into bits. Our approach is to use a single Hopf oscillator, then circuits of phase shift neurons to drive the legs and feet in the correct sequence.

So, let's see what we need to achieve. The literature reports the various gaits in a number of ways. For our hexapod there are three principal gaits: metachronal (wave), tetrapod (ripple), and tripod. One way of reporting these gaits is the phase lag of each leg. I use the left front leg as the reference (absolute phase is 0).



Perhaps the easiest to understand is the tripod gait; the legs move in three triplets, {LF, LR, RM} move together in one direction while {LM, RF, RR} move in the other direction, the second group phase-lagged by 180 degrees hence this group moves backwards while the first group moves forwards. Here's another way of representing the gaits.



Time is running to the right and the black bars show when the legs are raised, this is the 'swing' part of the legs' motion when the foot is not touching the ground. Again the tripod gait is easy to understand, the group {LF, LR, RM} all swing together with the feet raised while the group {LM, RF, RR} has feet on the ground. This is the 'stance' part of the legs' motion.

Our hexapod robot inhabits Webots[2], you can clearly see its legs and feet (they move up and down) in Fig.12.

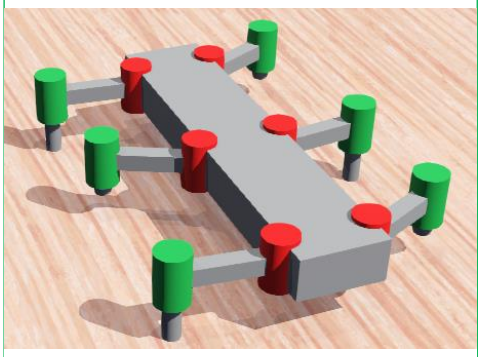
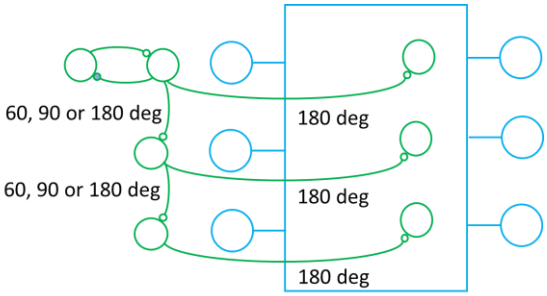


Figure 12 Hexapod robot showing legs and feet. It's executing a tripod gait

Now we must design our neural circuit to make these three types of gait happen. We shall use a single Hopf oscillator and drive the front-left (FL) foot and leg with the signal u_1. To drive the other legs, we need to shift the phase of this signal to obtain the phases shown in the diagram above. A glance at this diagram reveals two interesting facts. First the phase difference down the left side between LF, LM and between LM, LR is the same for a particular gait. The actual phase difference (60, 90, 180 degrees) depends on the gait. The second fact is that there is a 180 degree phase shift between contra-lateral limbs (LF-RF, LM-RM and LR-RR). This suggests the following architecture which is quite straightforward.

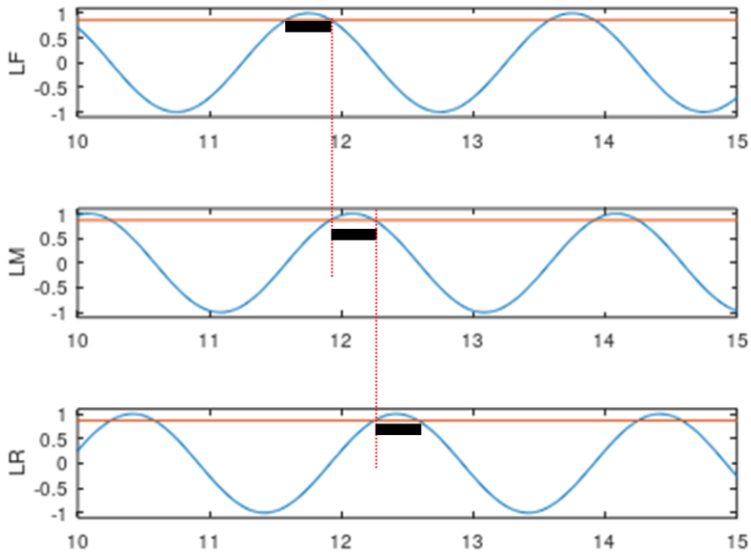


The blue circles represent the robot limbs (legs + feet) and the green circles neurons, or collections of neurons, since we need a chain of 3 neurons to produce a phase shift of 180 degrees. A constant phase shift is produced down the left neurons, then each left neuron is individually phase shifted

[2] It would be good to get this into Unreal-4

by 180 degrees and sent to the right. This is much simpler than the coupled oscillator approach used by others.

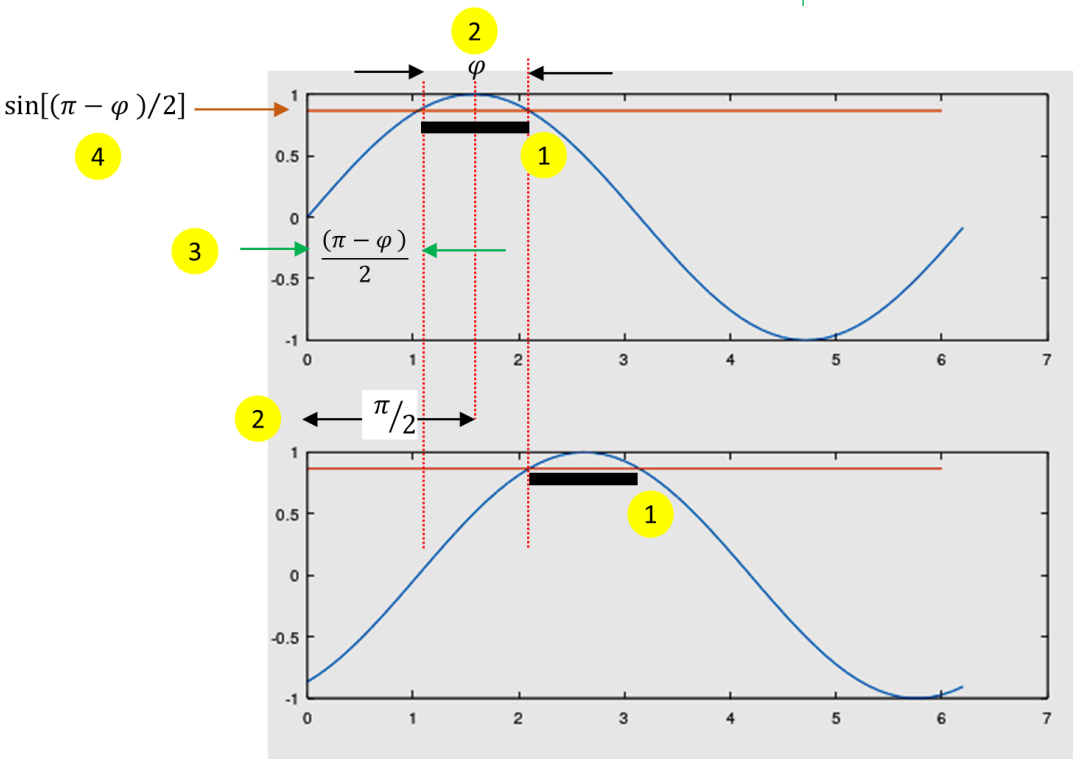
Let's have a look at some phase plots; we only need to consider the left limbs; the right phases are just mirrored. The diagram below shows a phase difference $\varphi = \pi/3$ (60 degrees) corresponding to the metachronal gait. The black bars show when the foot is lifted, as shown in the gait patterns presented above. So, let's discuss how to get the feet to move; we shall later return to the legs which are tricky buggers.



The black bars are correct and represent when signals must be presented to each foot to rise. You can see the red horizontal lines drawn at a particular threshold value, here that is 0.866 for our chosen phase lag value. Our code will tell each foot to rise when the phase is above this threshold. Easy! The only question is, how do we calculate this threshold value? Let's explore this with the help of the following diagram, where the stages in our argument are indicated by yellow blobs.

First (1) We draw the bars when the LF and LM feet are raised. Note that this happens when the phase (blue line) rises above the threshold (red line), which we are trying to

discover. Second (2) we identify the phase lag φ between the two feet and also we see the centre of the top sine wave is located at $\pi/2$. Then the important step (3) where we identify the angle where the foot starts to rise. Finally (4) we calculate the value of the threshold for this angle.



So, we have the expression for the threshold

$$sin[(\pi - \varphi)/2]$$

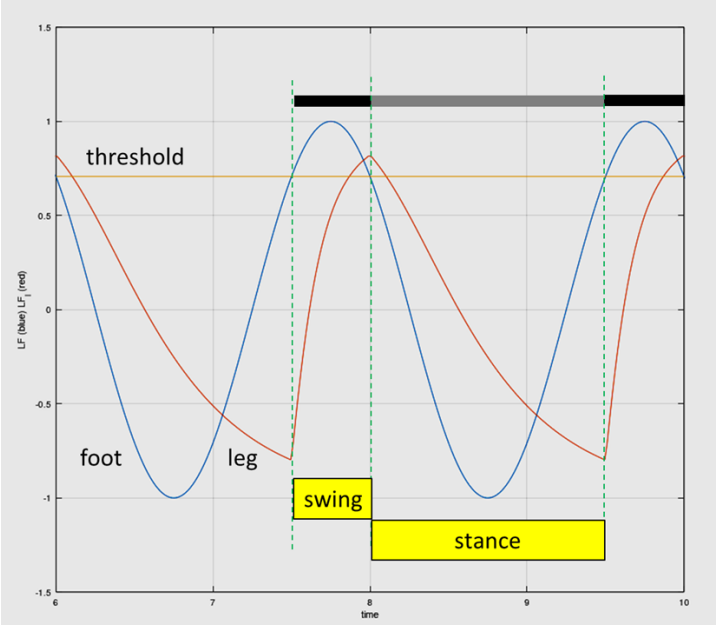
This is a great result since it is applicable for all three gaits with their own individual phase lags φ, and is easily translated into code, Fig.13.

```
double setFoot(double phase,
double thresh){
if(phase > thresh)
    return UP;
  else
    return DOWN;
}
```

Figure 13 Code snipped to raise and lower the feet

We now know how to get the feet working. The final piece of the puzzle is to get the legs moving correctly. The legs are driven by the same phases as the feet (before the threshold calculations). Legs have two parts to their motion, a *swing* when the feet are not in contact with the ground, and a *stance* where they are in contact with the ground and so provide

propulsion. So, legs and feet have to be coordinated in their motion. Let's have a look at the solution and then see how we got to it.



The blue curve shows the foot signal; when this rises above the threshold, the foot is up. So the leg must swing and this is the red signal which goes from negative (pointing backwards) to positive (pointing forwards). This signal comes from an additional neuron which receives input from the foot neuron *when it is above the threshold*. Here's the code for the additional neuron-17 which drive the LF leg.

```
if( y[0] > thresh)
  drive = 1;
else
  drive = 0;
dydt[17] = (-y[17] + 5.0*drive)/tau1;
```

That completes the development of our hexapod model, it only remains to provide the complete neural circuit.