Quadrupeds

Introduction

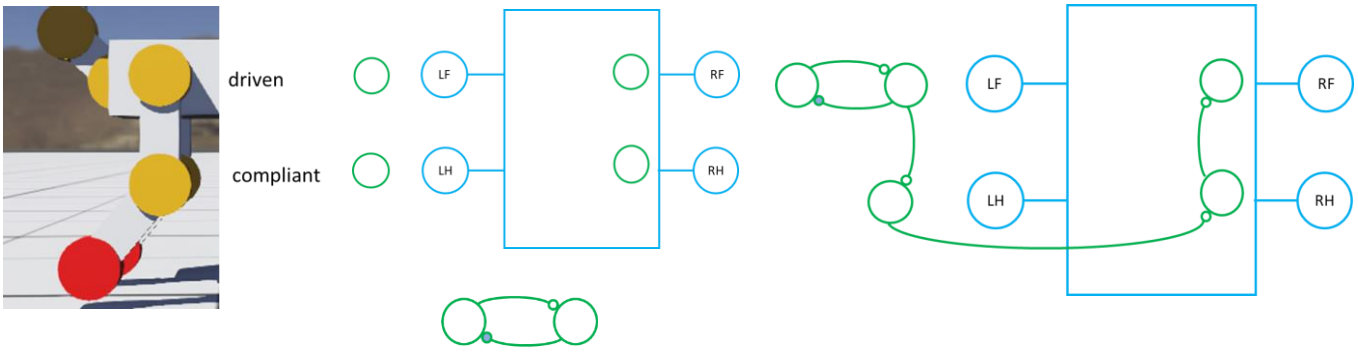
Quadrupeds (from mice to horses) display a number of gaits. Here is a great link to view some of these animated gaits https://vimeo.com/215637283 . If you look at this, or others, then you will see some fundamental facts about gaits: first, each leg swings periodically (it oscillates) and second, there is a phase delay between the legs. Finally, the set of phase differences are different for each gait!

So how could we model (and code) all of this? Well, the simplest way would be to directly specify the leg angles by using a sine wave !

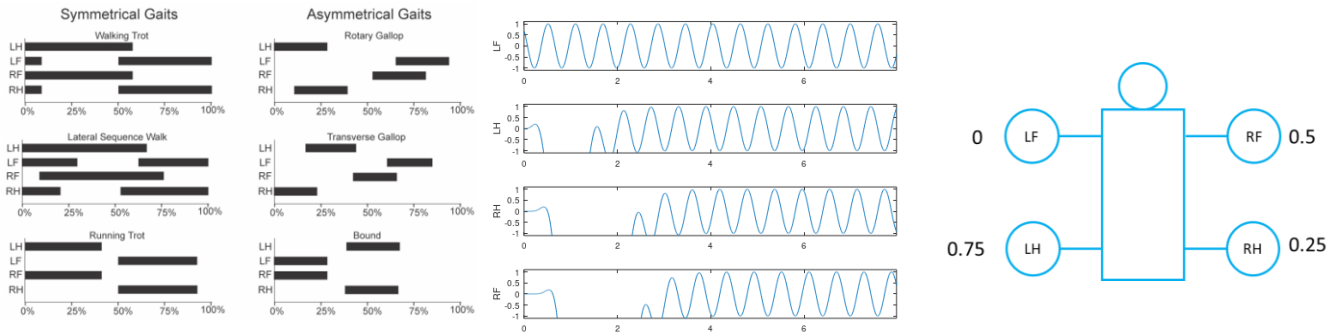
$$\theta_{leg(i)} = Asin(2\pi ft + \varphi_i)$$

where A and f are the amplitude and frequency of oscillation, and φ_i is the phase difference of the *i*'th leg relative to say the Left Front (LF) leg. You can see this in the 'ghostdog' sample robot: robots > epfl > biorob > ghostdog. But current robot research looks closely at real biological systems, quadruped gaits, snake or fish motion, object detection using neural circuits based on the fly-eye or bat-ear real biology. This is the approach we shall take here building a neural circuit to act as a Central Pattern Generator.

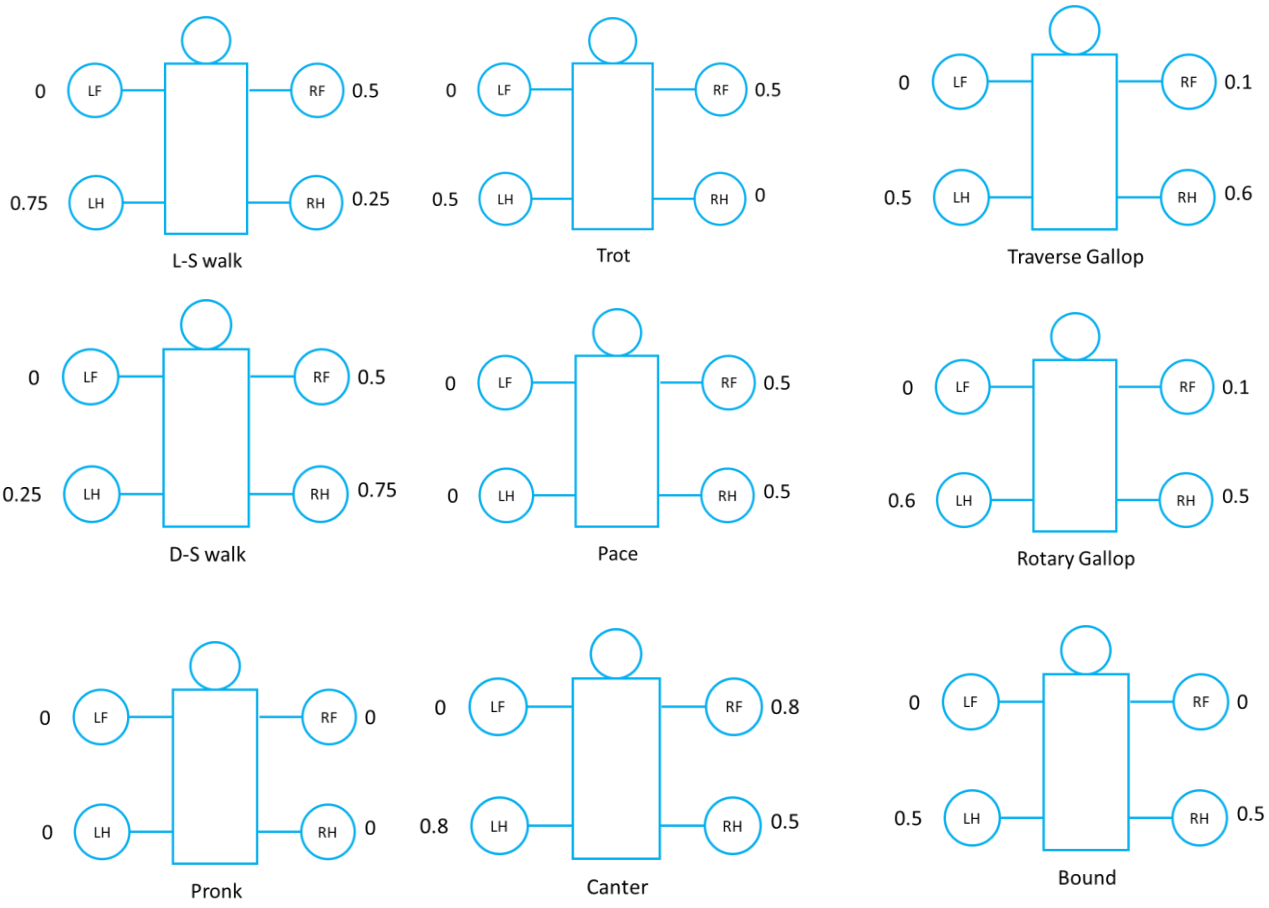
Let's build up a full CPG model, starting with the conceptual model. So what do we need? This is explored in the diagram below. We have four limbs and we are going to model the 4 hip joints. We shall assume that the knee joints are *compliant* which means that they act like springs, they can rotate but are not driven. You can see this on the left. Now to the neurons. We clearly need to drive each leg separately, so we must have a separate principal neuron, or a group of neurons) for each leg, shown in the middle. But we need oscillation, and this requires two coupled oscillators, bottom middle. Finally, we must connect the neurons together, to get the required phase difference between the legs. Here most researchers adopt a model which has a separate oscillator per leg (presented below), but my analysis has revealed a much simpler model, show below on the right, where only one oscillator is needed driving the front leg, and small groups of neurons provide the required phase difference.



The literature displays the various gaits, schematically in a number of ways; left contact with ground (black), middle joint angles (phases), right, phase difference of the legs



The phase difference plot perhaps the most useful in developing our model; they summarize succinctly experimental investigations of quadrupeds. Here's an assemblage of quad gaits with their phase differences. But what do the numbers 0.1, 0.5 etc mean? Well they are fractions of a complete circle or period. Take the **pace** gait for example. The LF and LH are in phase (swing together) and the RF and RH are both 0.5 out of phase. This means they are 0.5*360 degrees out of phase. So when LF and LH swing forward, RF and RH swing backwards.



Remember the numbers show the fraction of each phase difference (fraction of 360 degrees) relative to the LF leg. It turns out that we can classify these gaits according to their speeds. Low-speed gaits are L-S walk and D-S walk, middle speeds are Trot and Pace and high speeds are Bound, Transverse and Rotary Gallops. Unsure about Pronk and Canter.Model – From maths to code (my research)

The Mathematical Model (my research)

The CPG model is expressed as a series of ordinary differential equations (ODEs) which are passed to a C-code solver routine **rkf45**. You will code the ODEs in the right-hand-side file **CBP_Quadruped_2_rhs.c** (or your renamed copy). Let's look at the maths and then turn to the coding.

First the neuron pair forming the oscillator. The ODEs for the neurons are,

$$\frac{dy_0}{dt} = g\left(\frac{r_0}{r} - 1\right)y_0 - \omega y_1$$

$$\frac{dy_1}{dt} = \omega y_0 + g\left(\frac{r_0}{r} - 1\right)y_1$$

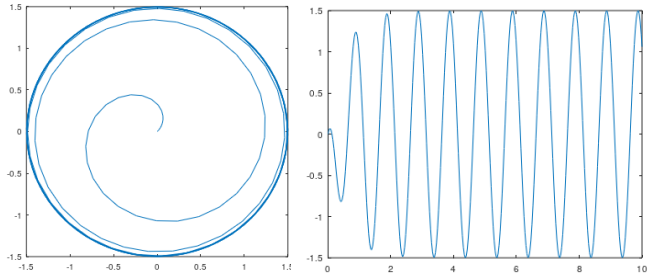
where

$$r = \sqrt{y_0^2 + y_1^2}$$

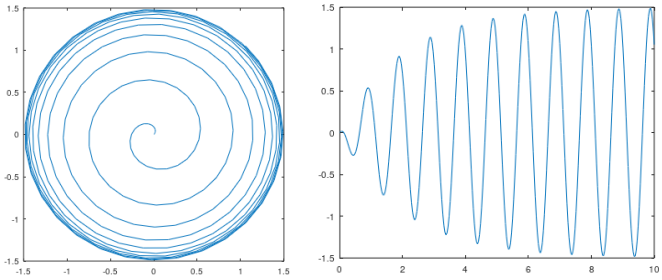
The principal variables of the oscillator are y_0 and y_1. Here's a table of the parameters and their meaning

Math	Code	Meaning
g	g	Rate at which oscillator approaches the limit cycle.
r_0	r0	Radius of oscillator limit cycle
ω	omega	Frequency of oscillation in radians per second. Linked to frequency in Hertz by $\omega = 2\pi f$

The diagram below should help you understand the meaning of the parameters g and r0. You can see the phase plane and the plot of y[0] versus time for a couple of values of g. First, note that the amplitude of y[0], and the radius of the limit cycle both approach the value of g = 1.5. Second, note that the larger value of g on the left means that the solution approaches the limit cycle more rapidly.



r0 = 1.5, g = 2.0



r0 = 1.5, g = 0.5

Now to the neural circuits that link from LF to the other legs. These circuits are made from 4 neurons. Each obeys the equation (here shown for the n'th neuron)

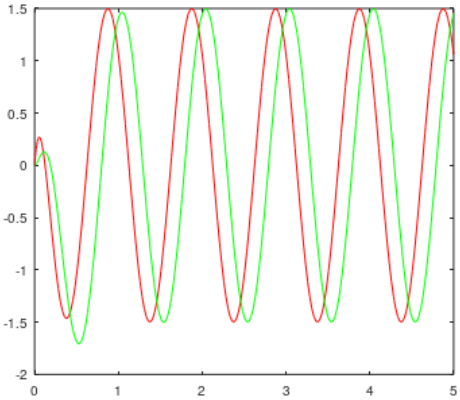
$$\frac{dy_n}{dt} = \frac{1}{\tau}(-y_n + kI)$$

The time constant τ determines how fast the neuron responds to the input *I* and *k* is a constant that multiplies the input. Theory (see Appendix A) determines how to set these constants for a desired phase difference. It turns out that if the required phase difference is φ between the input *I* and the output y_n then we must have,

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

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

The diagram below shows the behaviour of a single phase-shift neuron.



The red curve shows the sine wave input to a single neuron, and the green curve its output. The requested phase difference was 60-degrees. You should be able to see that the phase delay is 1/6 of a complete cycle of 360-degrees which is 60! Also note that the output amplitude is the same as the input. This is the result of the calculation of **tau** and **k** according to the expressions given above. So the green leg follows the red leg.

Now there is a limitation. Theory proves that a single neuron can only give less than a 90-degree phase shift. So what happens if we need a larger phase delay? We can solve this by chaining a number of neurons and dividing the phase shift over the chain. Here is a chain of 4 neurons where the input is from y_0 (which is part of the oscillator pair).

$$\frac{dy_2}{dt} = \frac{1}{\tau}(-y_2 + ky_0)$$

$$\frac{dy_3}{dt} = \frac{1}{\tau}(-y_3 + ky_2)$$

$$\frac{dy_4}{dt} = \frac{1}{\tau}(-y_4 + ky_3)$$

$$\frac{dy_5}{dt} = \frac{1}{\tau}(-y_5 + ky_4)$$

So if our required phase difference is φ then we must set the constants to,

$$\tau = \frac{\tan \varphi/4}{\omega}$$

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

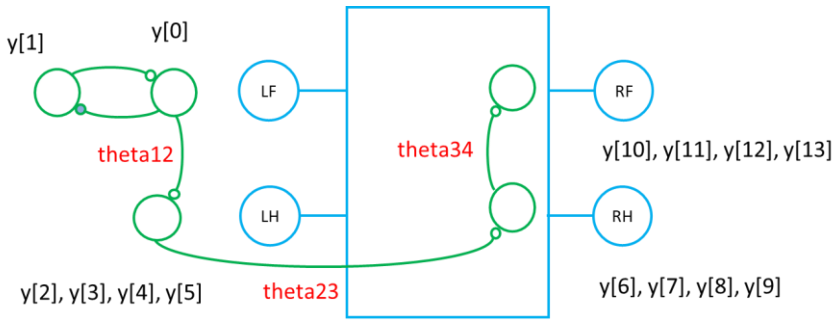
See how the phase difference per neuron is the desired phase difference divided by 4.

Coding the Model

Code in CBP_Quadruped_2_rhs.c

Most of the code will be added to **CBP_Quadruped_2_rhs.c** The following diagram shows the labelling of the neurons in the code. To code the math you just need to remember the following correspondances

Math	**Code**
y_n	**y[n]**
$\frac{dy_n}{dt}$	**dydt[n]**
k	**k**
τ	**tau**
ω	**omega**



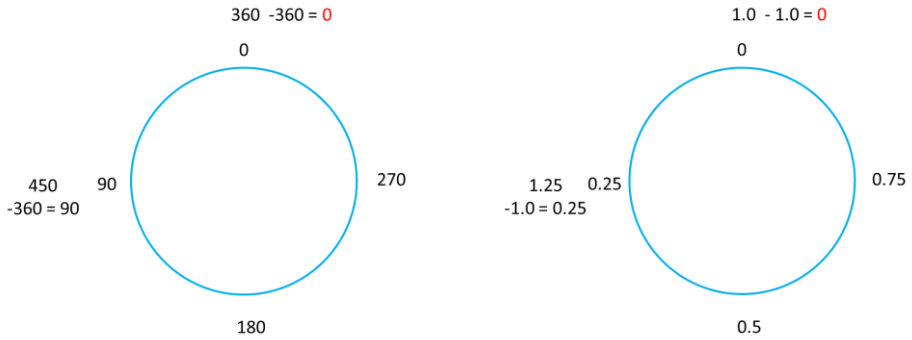
The array variables used above appear in the code. The phase differences in red are passed though from the calling code **CBP_Quadruped_2.c** to the solver using the function **setParams(…);**

So to code this model, work down the oscillator chain, converting each ODE to a line of code. Once you have the oscillator and first group of 4 coded, then the rest just drops in. Only thing to remember is to specify the phase before each group of 4. So for the first group you would write

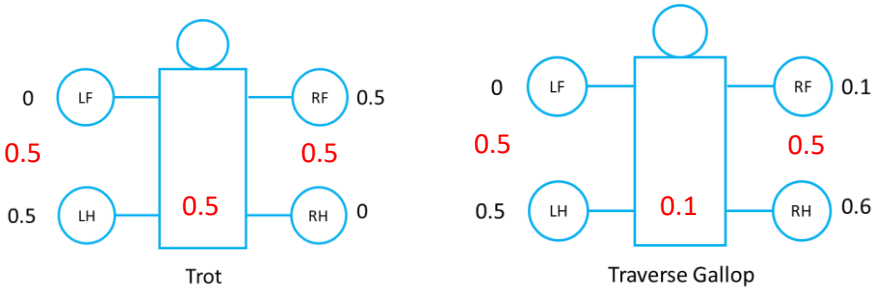
```
// Left front to Left hind
phase = theta12;
tau = tan(phase/4.0)/omega;
k = sqrt(tau*tau*omega*omega + 1);
```

Coding in CBP_Quadruped_2.c

Much of the grunt-work is already done. Note that the program takes a command line variable (set in the scene tree) which selects which gait is selected. The numbers are in the #defines and used in the switch block. Each case sets the phase shifts between the legs; theta12, theta23, theta34. To do this, we need a nifty trick which involves numbers on a circle. Let's step around a circle in steps (phase increments) of 90-degs. So we go 0, 90, 180, 270, 360, 450. But wait, 360 is the same as 0-deg and 450 is the same as 90 deg. Rule is, if in our calculations (or code) any angle greater than 360, then we must subtract 360. The same is true when we work in fractions of going around the circle, bottom right



So let's apply this to a couple of gaits and you can do the rest. Let's start with the trot, working around the legs from LF. To get LH to be at 0.5, we add 0.5 (red). To get RH to be 0 we add another 0.5 (red) since 0.5 + 50. = 0 (takeaway 1). To get RF to be 0.5 we add another 0.5 (red). So all phase differences are 0.5.



The transverse gallop starts the same, we add 0.5 (red) to LF to get LH to be 0.5. Then to get RH to be 0.6 we need to add on 0.1 (red). Now here comes the trick – to go from RH = 0.6 to RF which is 0.1, we add on 0.5 (red), since 0.6 + 0.5 = 1.1, but we subtract 1.0 to give us 0.1. So here are the phase differences you will need to put in your case statement for the above examples,

Trot	
theta12	0.5*2.0*M_PI
theta23	0.5*2.0*M_PI
theta34	0.5*2.0*M_PI

Gallop	
theta12	0.5*2.0*M_PI
theta23	0.1*2.0*M_PI
theta34	0.5*2.0*M_PI

One final thing. You cant have 0.0 for a phase difference, since tan(0) is 0 which would make tau = 0 and dividing by this (rhs code) won't be pleasant. So instead of 0.0, use 0.001.

Finally (really) all CPG models need to convert the neuron outputs into leg positions (angles). This is a whole topic in itself. For this model we use the approach used in the Webots sample model robots > epfl > biorob > ghostdog. Here the phases are used to directly drive the legs, but their values are scaled and an offset is added to obtain the standing position. These are tuned by hand, so you will have to do this.

Investigations

There's so many possibilities! A canny researcher would make some initial short investigations, and see if anything interesting emerges. They would then choose one investigation to push as far as they can.

1) Have you been able to reproduce any of the gaits? How well did your model work? Did you manage to get values for scaling/H and offset/H that work for all gaits? Use the movie-clip to evaluate the performance of your model https://vimeo.com/215637283

2) Perhaps code one or more gaits using the procedural approach and compare your CFG model's gait with the procedural gait

3) Measure the speeds of the various gates (after you learn to spell gaits correctly)

4) Investigate the effect of parameters, (**omega, r0, etc)** e.g., on the speed of one or more gaits.