# Central Pattern Generator (CPG) document

## Central Pattern Generator

图示

描述已自动生成

The CPG model is a dynamic system used to generate a series of sine waves and make the phase difference, frequency, and amplitude of these sine waves adjustable in real time. You can change the parameter of CPG at any time, and this will result the output sinusoidal waves to change, but such transition will be continuous in value. This article includes a CPG model written in python for ease of use. CPG enables the sine wave to produce a continuous transformation curve when switching amplitude, frequency, phase and offset. CPG can be described as

Where and are internal states of CPG, is the number of output sine wave channels.

Below is an example of A and B matrix



Let be the channel index. are parameters to control the rate of convergence. , , and are inputs of CPG. controls the amplitudes of the channels, are the angular frequencies of the channels, are the offsets of the channels and are the phase difference of the channels. (the order of indices is important!). For example, setting

will result two output channels by , where

is determined by the initial value of .

## Performance

Here shows the output of CPG tracking 10 channels of sinusoidal waves, with given frequency, amplitude and phase difference. Notice that CPG only tracks the PHASE DIFFERENCE between neighboring channels instead of the absolute phase, which means the CPG output can have a unknown phase shift compared with the desired signal.

图表, 散点图

描述已自动生成

The phase shift error through time

图形用户界面

描述已自动生成

The amplitude error through time:

图示, 工程绘图

描述已自动生成

## Python Implementation

The class written in Python is CPG()

Arguments:

|  |  |  |
| --- | --- | --- |
| variables | meaning | Data type |
| n | Number of channels | Positive integer |
| R | Target Amplitudes | Numpy array of size (n, 1) |
| omega | Target Angular frequencies | Numpy array of size (n, 1) |
| mu | As the goal of CPG is to track the given frequency, amplitude and phase difference, ‘mu’ controls the phase convergence rate in phase tracking. | Numpy array of size (n, 1) |
| a | As the goal of CPG is to track the given frequency, amplitude and phase difference, ‘a controls the amplitude convergence rate in phase tracking. | Numpy array of size (n, 1) |
| dp | Desired phase shift between channels , | Numpy array of size (n-1, 1) |
| Init\_phi | Initial values of | Numpy array of size (n, 1) |
| offset | Offsets | Numpy array of size (n, 1) |
| dt | Simulation step, for example simulation by 200Hz results dt=0.005 | float |
| lower\_lim | Per channel lower bound, the output of CPG lower than this value will be cut. | Numpy array of size (n, 1) |
| upper\_lim | Per channel upper bound, the output of CPG higher than this value will be cut. | Numpy array of size (n, 1) |

Where typical choice of ‘mu’ is 1.0 and typical choice of ‘a’ is 1.0. Here shows an example of initialization of a CPG with 200Hz (period 0.005s) update.

n\_joints = 5

cpg = CPG(n=n\_joints,

R=np.array([[1.0], [1.0], [1.0], [1.0], [1.0]]),

omega=np.array([[0.1], [0.1], [0.1], [0.1], [0.1]]),

mu=np.array([[1.0], [1.0], [1.0], [1.0], [1.0]]),

a=np.array([[1.0], [1.0], [1.0], [1.0], [1.0]]),

dp=np.array([[math.pi/5], [math.pi/5], [math.pi/5], [math.pi/5]]),

init\_phi=np.zeros((n\_joints, 1)),

offset=np.zeros((n\_joints, 1)),

dt=0.005,

lower\_lim=-2 \* np.ones((n\_joints, 1)),

upper\_lim=2 \* np.ones((n\_joints, 1)))

The member functions are

set\_amp(R)

Function: set new amplitudes

Input:

|  |  |  |
| --- | --- | --- |
| R | Target amplitudes | Numpy array of size (n, 1) |

Output: None

set\_freq(omega)

Function: set new frequencies

Input:

|  |  |  |
| --- | --- | --- |
| omega | Target angular frequencies | Numpy array of size (n, 1) |

Output: None

set\_phase\_shift(dp)

Function: set new phase shift between any two neighboring channels

Input:

|  |  |  |
| --- | --- | --- |
| dp | Desired phase shift between channels , | Numpy array of size (n-1, 1) |

Output: None

update()

Function: evolve the internal states to the next time step. For example, you are controlling a system with 50Hz, you have to call this function at each step.

Input: None

Output: None

reset(init\_phi)

Function: reset the CPG to initial state

Input:

|  |  |  |
| --- | --- | --- |
| Init\_phi | Initial values of | Numpy array of size (n, 1) |

Output: None

output()

Function: output sine waves values at current time step

Input:

|  |  |  |
| --- | --- | --- |
| Init\_phi | Initial values of | Numpy array of size (n, 1) |

Output: None

A typical use of CPG is

n\_joints = 5

cpg = CPG(n=n\_joints,

R=np.array([[1.0], [1.0], [1.0], [1.0], [1.0]]),

omega=np.array([[0.1], [0.1], [0.1], [0.1], [0.1]]),

mu=np.array([[0.5], [0.5], [0.5], [0.5], [0.5]]),

a=np.array([[0.1], [0.1], [0.1], [0.1], [0.1]]),

dp=np.array([[math.pi/5], [math.pi/5], [math.pi/5], [math.pi/5]]),

init\_phi=np.zeros((n\_joints, 1)),

offset=np.zeros((n\_joints, 1)),

dt=0.05,

lower\_lim =np.zeros((n\_joints, 1))-1,

upper\_lim =np.zeros((n\_joints, 1))+1)

for i in range(200):

cpg.update()

x = cpg.output()

Then you can use cpg to control external systems

By setting the parameters of CPG, one can get different performances of sine waves

Phase shift by , below is the plot of output x

图片包含 图示

描述已自动生成

You can change the amplitudes in midway of simulation by set\_amp() function:

图片包含 图示

描述已自动生成

Or changing frequency by either set\_freq() function

图片包含 图示

描述已自动生成

Phase shift from to to

图片包含 形状

描述已自动生成

CPG will always give a continuous output.