# NISE tutorial on CPG

## 22.10.2024

**Dr.Ing.habil. John Nassour**

**[Goal]**

Today's session aims to learn how to program a central pattern generator in the open-source Arduino software using C language and upload it to a microcontroller ESP32. We will use https://wokwi.com to simulate the ESP32.

Central pattern generators use differential equations that need to be solved. The solution of these equations is not always simple to find using a classical way of integration; therefore, numerical methods come into account.

**[Understanding by an example]**

In the attached file "**RS_neuron.ino**" you find the C code to program the pattern generator used in my paper [Nassour et al. 2014]. The paper uses a neural model introduced in [Rowat and Selverston 1997]. It consists of a system of two first-order dynamic equations:

$$\tau_m . \frac{dV}{dt} = -(fast(V, \sigma_f) + q - i_{\text{inj}})$$

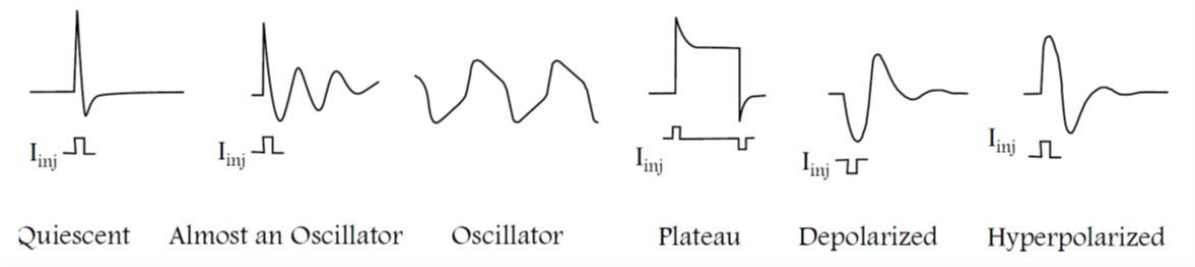
$$\tau_s . \frac{dq}{dt} = -q + q_\infty(V)$$

Where:

$$\text{fast}(V, \sigma_f) = V - A_f \tanh((\sigma_f / A_f)V)$$

$$q_\infty(V) = \sigma_s(V - E_s)$$

Changing the conductivities "**sigma_s**" and "**sigma_f**" changes the nature of the generated pattern.




The provided code (**RS_neuron.ino**) details the modulation of the neural structure with this system of two ordinary differential equations.

First of all, we create a structure of our neuron model containing all variables:

```
/*******************************************************/
//struct RSneuron
/*******************************************************/
struct RSneuron {
   char discription[DISCRIPTION_LENGTH]; // name
   double tao_m   = 0.1;
   double tao_s   = 20*tao_m;
   double Af      = 5;
   double Es      = 0;
   double sigma_s = 0;
   double sigma_f = 0;
   double V_0     = 0;
   double q_0     = 0;
   double V       = V_0;
   double q       = q_0;
   double inj_cur = 0;
   double inj_cur_MultiplicationFactor = 1;

} rs_neuron[NOMBER_RS_NEURONS];
/*******************************************************/
```

Then we create a structure to configure the patterns; it only contains intrinsic variables that make a major difference in the pattern. In addition to the conductivities "**sigma_s**" and "**sigma_f**", we include the time constant "tao_m" which represents the dynamics of the system, and a multiplication factor for the injected current of the neuron that represents the amplitude of the quiescent pattern.

```
/*******************************************************/
//struct Pattern
/*******************************************************/
struct Pattern{
  double sigma_s;
  double sigma_f;
  double tao_m;
  double InjCurrentMultiplicationFactor;
  };
/*******************************************************/
```

Four types of patterns are listed afterward: OSCILLATORY, QUIESCENT, PLATEAU, and ALMOSTOSC.

Then we write the function "**update_RS_neuron()**" that will be called repeatedly every time step to calculate the solution at that particular time. We used "**Runge–Kutta methods**" to calculate the solutions of both differential equations presented above. Since we have two equations to solve, we used "**k1, k2, k3, k4**" to indicate the slops in the first equation and "**l1, l2, l3, l4**" for the slopes in the second equation.

Please notice that this function will be called repeatedly, and therefore, the initial values are set before the first call.

```
/*****************************************************/
void update_RS_neuron(struct RSneuron* rs_n)
{
int n = 20;
//double x0 = t-ts;
//double xf = t;
double h;
//h = (xf-x0)/(double)n;
h= ((double)mydelay/1000)/n;
double xa[20],ya[20],yb[20];
double k1,k2,k3,k4,k, l1,l2,l3,l4,l;
//xa[0] = x0;
ya[0] = rs_n->V;
yb[0] = rs_n->q;

for (int i=0; i<n-1 ; i++)
{
  k1= h*fun_v( ya[i] , yb[i] , rs_n->sigma_f , rs_n->inj_cur*rs_n->inj_cur_MultiplicationFactor ,  rs_n->tao_m , rs_n->Af );
  l1 = h*fun_q( ya[i] , yb[i] , rs_n->tao_s , rs_n->Es , rs_n->sigma_s );

  k2= h*fun_v( ya[i]+k1/2 , yb[i]+l1/2 , rs_n->sigma_f , rs_n->inj_cur*rs_n->inj_cur_MultiplicationFactor , rs_n->tao_m , rs_n->Af );
  l2 = h*fun_q( ya[i]+k1/2 , yb[i]+l1/2 , rs_n->tao_s , rs_n->Es , rs_n->sigma_s);

  k3= h*fun_v( ya[i]+k2/2 , yb[i]+l2/2 , rs_n->sigma_f , rs_n->inj_cur*rs_n->inj_cur_MultiplicationFactor , rs_n->tao_m , rs_n->Af );
  l3 = h*fun_q( ya[i]+k2/2 , yb[i]+l2/2 , rs_n->tao_s , rs_n->Es , rs_n->sigma_s );

  k4= h*fun_v( ya[i]+k3 , yb[i]+l3 , rs_n->sigma_f , rs_n->inj_cur*rs_n->inj_cur_MultiplicationFactor , rs_n->tao_m , rs_n->Af );
  l4 = h*fun_q( ya[i]+k3 , yb[i]+l3 , rs_n->tao_s , rs_n->Es , rs_n->sigma_s );

  k = 1/6.0 * (k1 + 2*k2 + 2*k3 + k4);
  l  = 1/6.0 * ( l1 + 2*l2  + 2*l3  +  l4);

  ya[i+1]= ya[i]+k;
  yb[i+1]= yb[i]+l ;
  //xa[i+1]= xa[i]+h;
}
rs_n->V = ya[n-1];
rs_n->q = yb[n-1];

return;
}
```

The function "**setup_RS_neurons()**" is used to configure a neuron with a different configuration of those assigned by default.

```
/*****************************************************/
void setup_RS_neurons(struct RSneuron* rs_n,struct Pattern myP, String str)
{
for(int i=0 ; i<(sizeof(str) / sizeof(str[0])) ; i++)
     rs_n->discription[i] = str[i];

rs_n->tao_m = myP.tao_m;
rs_n->tao_s = 20 * myP.tao_m;
rs_n->sigma_s = myP.sigma_s;
rs_n->sigma_f = myP.sigma_f;
rs_n->inj_cur_MultiplicationFactor = myP.InjCurrentMultiplicationFactor;

}
/*****************************************************/
```

The function "**update_locomotion_network()**" will be called repeatedly and it hosts all updates required in the central pattern generator network. In the current example, we only update the pattern generation neurons "RS_neuron". However, with additional neural structures involved (such as sensory neurons, motor neurons, and pattern formation neurons), all updates occur in this function. In the current example, we only update two RS_neurons:

```
/*******************************************************/
void update_locomotion_network(void)
{
for (int i = 0; i< NOMBER_RS_NEURONS ; i++)
  update_RS_neuron(&rs_neuron[i]);
}
/*******************************************************/
```

In the main loop function, we injected two currents for both neurons at different times. Injecting current is essential to produce a pattern on the “RS_neuron”.

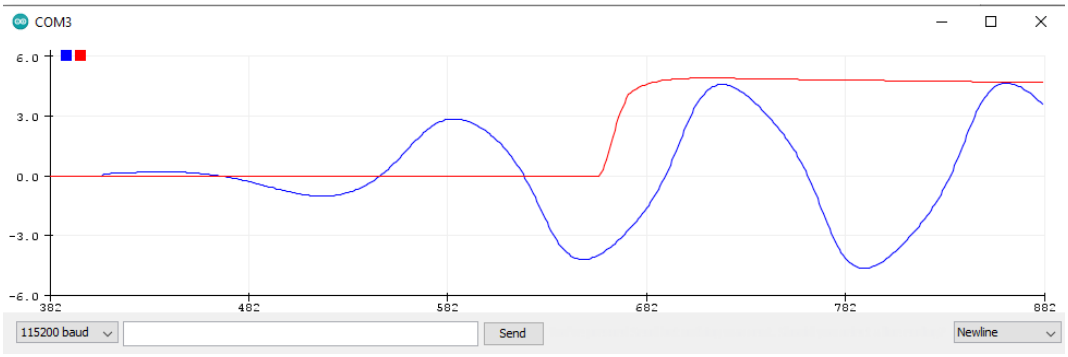
```
/* After 5 seconds, inject a current in the first neuron for a duration of 0.01 second*/
if((myTime>5000)&&(myTime<5010))
rs_neuron[0].inj_cur = 1;
else
rs_neuron[0].inj_cur = 0;

/* After 8 seconds, inject a current in the second neuron for a duration of 0.2 second*/
if((myTime>8000)&&(myTime<8200))
rs_neuron[1].inj_cur = 1;
else
rs_neuron[1].inj_cur = 0;
```

To demonstrate the activity of the neurons, we used “**Serial.print**” function to stream the values on the serial port and plot it by the “Serial Plotter “ from the “tools” menu in the Arduino SDK.

```
/* Printing the output of the neurons on serial port*/
for (int i = 0; i< NOMBER_RS_NEURONS ; i++)
{Serial.print(rs_neuron[i].V);Serial.print(" ");}
Serial.print("\n");
```

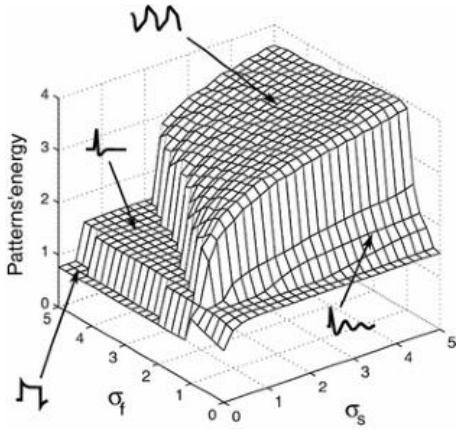
Please see the output:




**[Your task]**

1. In the example “**RS_neuron.ino**” add two additional neurons to the network. And demonstrate four different patterns (OSCILLATORY, QUIESCENT, PLATEAU, and ALMOSTOSC), each neuron with a single pattern. **[10% of the grade]**

2. By trial and error, find the oscillatory pattern that approximately doubles the frequency of the oscillatory pattern presented in the example. Demonstrate your results on two neurons using two different patterns. The finger presented below gives you some hints to understand how patterns are distributed in the space of "sigma_s" and "sigma_f" **[10% of the grade]**




3. Refer to the paper [Matsuoka 1985] to modulate the Mutually Inhibiting Neurons presented in section 2 of the paper, please use the adaptation terms as well. Demonstrate Figure 1 on the step response with the three different cases. **[20% of the grade]**. Demonstrate the coupling of two mutually inhibiting neurons as in Figure 2 **[20% of the grade]**.
4. Select different coupling on three and four neurons and demonstrate the output. **[10% of the grade]**.
5. Inspired by Figure 1 in the paper [Ijspeert et al. 2007], propose a coupling for ten motors snake using 20 neurons. **[10% of the grade]**, use Matsuoka oscilator.
6. Optimize your solution method of the differential equations so that you can obtain your program of step 5 running faster, and report the cycle time **[20% of the grade].**

Each step should be submitted in a single code that can be tested by the tutor on the ESP32 (wokwi.com), the output is streamed on the serial port and presented on the serial plotter.

**Please also submit a text document (PDF) indicating the coupling, main keys of the work, and the outputs as a screenshot of the "serial plotter" if applicable.**

The deadline is **(see course page on moodle)**.

**References:**

[Nassour et al. 2014] Multi-layered multi-pattern CPG for adaptive locomotion of humanoid robots. J Nassour, P Hénaff, FB Ouezdou, G Cheng. Biological cybernetics 108 (3), 291-303.

[Rowat and Selverston 1997] Rowat P.F., Selverston, A.I., Oscillatory Mechanisms in Pairs of Neurons Connected with Fast Inhibitory Synapses, Journal of Computational Neuroscience, 1997.

[Matsuoka 1985] Matsuoka, K. Sustained oscillations generated by mutually inhibiting neurons with adaptation. Biol. Cybern. 52, 367–376 (1985). https://doi.org/10.1007/BF00449593

[Ijspeert et al. 2007] Ijspeert AJ, Crespi A, Ryczko D, Cabelguen JM. From swimming to walking with a salamander robot driven by a spinal cord model. Science. 2007 Mar 9;315(5817):1416-20. doi: 10.1126/science.1138353. PMID: 17347441.