Student names: Genilloud Florian, Emilio Fernandez, Joachim Dunant

Swimming with Salamandra robotica – CPG Model

In this project you will control a salamander-like robot Salamandra robotica II for which you will use Python and the PyBullet physics engine. Now you have an opportunity to use what you've learned until now to make the robot swim and eventually walk. In order to do this, you should implement a CPG based swimming controller, similarly to the architecture shown in Figure 1.

The project is based on the research of [1], [2] and [3]. It is strongly recommended to review [3] and its supplementary material provided on the Moodle. You will be tasked with replicating and studying the Central Pattern Generator (CPG).

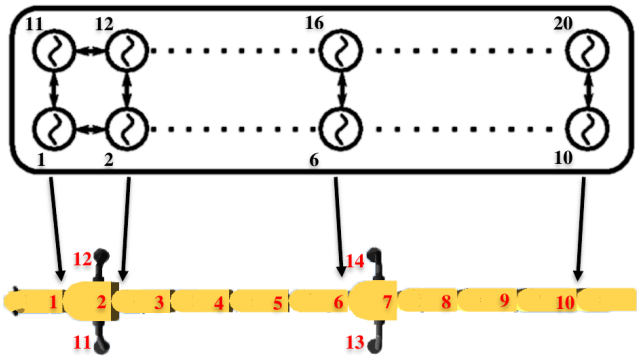


Figure 1: A double chain of oscillators controlling the robot's spine.

Fun fact: The salamander's name is T-Rex V.3141592

8a. Implement a double chain of oscillators along with limb CPG's

Salamandra robotica has 10 joints along its spine and 1 joint for each limb. The controller is defined as

$$\dot{\theta}_i = 2\pi f + \sum_j r_j w_{ij} sin(\theta_j - \theta_i - \phi_{ij}) \tag{1}$$

$$\dot{r}_i = a(R_i - r_i) \tag{2}$$

$$q_i = r_i(1 + cos(\theta_i)) - r_{i+10}(1 + cos(\theta_{i+10})) \text{ if body joint} \tag{3}$$

with θ_i the oscillator phase, f the frequency, w_{ij} the coupling weights, ϕ_{ij} the nominal phase lag (phase bias), r_i the oscillator amplitude, R_i the nominal amplitude, a the convergence factor and q_i the spinal joint angles. For more information, please refer to [3]. Also note how these are the same equations, although Equation (2) has been simplified into a first order ODE in order to simplify the implementation in this project.

1. We have implemented the double chain oscillator model by including equations 1 and 2 in the function `network.py::network_ode`. The parameters appearing in those equations are defined in `robot_parameters.py` and in `simulation_parameters.pys` using the following information obtained from the supporting material for [3]:

	Body oscillator	Limb oscillator
Number of oscillators	16	4
a_i in 1/s	20.0	20.0
$[d_{low}, d_{high}]$ in arbitrary units	[1.0, 5.0]	[1.0, 3.0]
$[c_{\nu,1}, c_{\nu,0}]$ in Hz	[0.2, 0.3]	[0.2, 0.0]
$[c_{R,1}, c_{R,0}]$ in radians	[0.065, 0.196]	[0.131, 0.131]
ν_{sat} in Hz	0.0	0.0
R_{sat} in radians	0.0	0.0
$[w_{ij}, \phi_{ij}]$ downwards in body CPG	[10.0, -2π/8]	
$[w_{ij}, \phi_{ij}]$ upwards in body CPG	[10.0, 2π/8]	
$[w_{ij}, \phi_{ij}]$ contralateral in body CPG	[10.0, π]	
$[w_{ij}, \phi_{ij}]$ from limb to body CPG	[30.0, π]	
$[w_{ij}, \phi_{ij}]$ within the limb CPG		[10.0, π]

Figure 2: Parameters for CPG model

$$\nu = g_\nu(d) = \begin{cases} c_{\nu,1}\, d + c_{\nu,0} & if \quad d_{low} \le d \le d_{high} \\ \nu_{sat} & otherwise \end{cases}$$

$$R = g_R(d) = \begin{cases} c_{R,1}\, d + c_{R,0} & if \quad d_{low} \le d \le d_{high} \\ R_{sat} & otherwise \end{cases}$$

Figure 3: Saturation functions for frequencies and nominal amplitudes

The weights and phase bias are set with a 24x24 matrix after reading a csv file defined as described in 2. The rest of the parameters are used for defining the saturation functions in 3. Finally, we test out implementation by running the network using `run_network.py`.

2. We implement the output of your CPG network in the function `network.py::motor_output` to generate the body spinal joint angles according to equation 3. For the phase spinal joints we simply take the value of the phase if the amplitude is non-zero

3. Finally, we implement an increasing drive in order to show that our network can generate swimming and walking patterns similarly to [3]. We will know compare our resulting plots with the "model" plots in 4 and 6 obtained from [3]

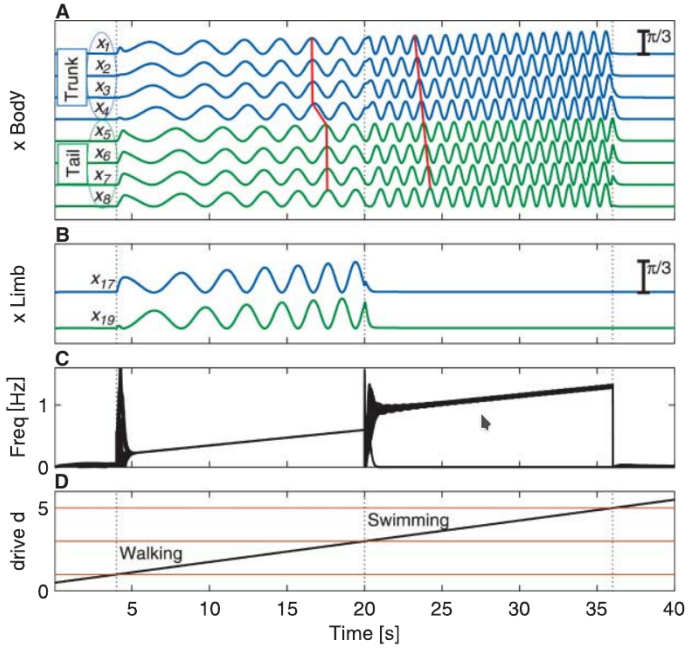
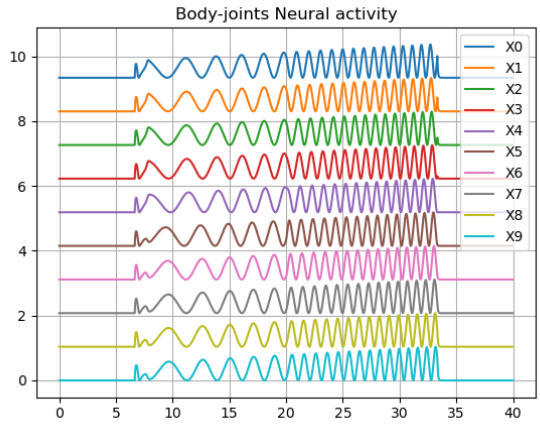
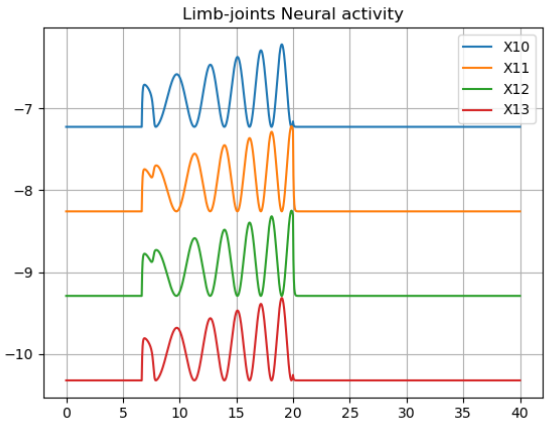


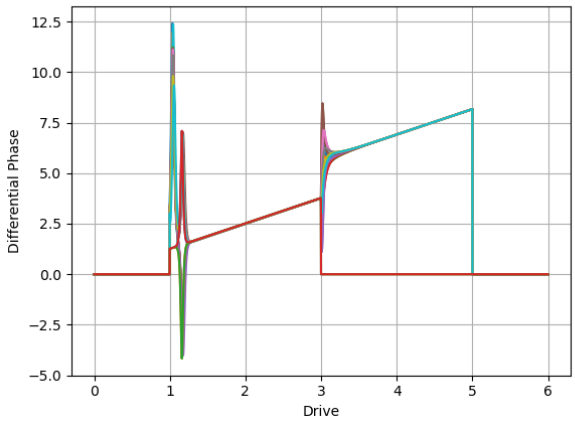
Figure 4: Oscillator patterns from [3]



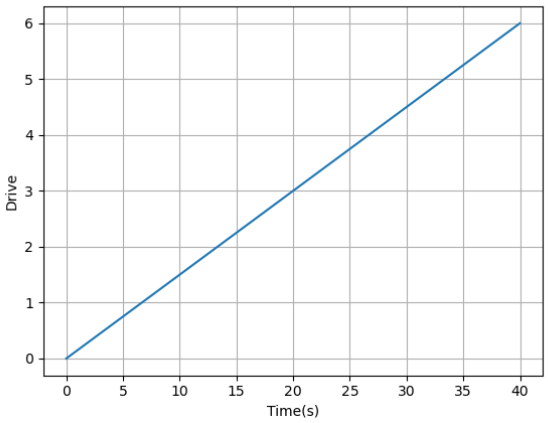
(a) Body joints output



(b) Limb joints output



(c) Frequency



(d) Drive

Figure 5: Oscillation patterns from our network (compare with Fig. 4)

Oscillation patterns comparison: Above we have the oscillation patterns generated from our network. First of all, we observe that the drive (Figure 5d is identical to the one shown in 4.D: it increases from 0 to 6 during 40 seconds, so that the salamander can walk (after 4 secs) and swim (after 20 secs). Figure 5a and 5b are obtained using the equation $q_i = r_i(1 + cos(\theta_i))$, which corresponds to the output neural activities for all the activities. They behave as expected: during walking we observe a standing wave with long cycle duration, whereas during swimming we observe a shorter-cycle duration travelling wave in body joints and the limb joints are tonic. Finally, Figure 5c shows the differential phases (in radians). If we compare it with the "model" plot (4.C we observe that they have a similar shape: there's a burst when the salamander starts walking as well as when the salamander starts swimming. Probably the first burst could be improved in our network, there might be a problem with the motor outputs at the beginning of the simulation. In general, we can conclude that we have successfully reproduced the oscillation patterns from [3].

Oscillation properties comparison: We have already verify the correct functioning of our network. Nonetheless, we also want to compare the oscillation properties of our network, especially the saturation functions shown in Figure 7. We observe that their behavior is identical to Figure 6(A and B). The body joints are "active" during both walking and swimming whereas the limb joints are only active during walking because they saturate during swimming. We also see that limb joints oscillate with lower frequency but with higher amplitude than body joints.

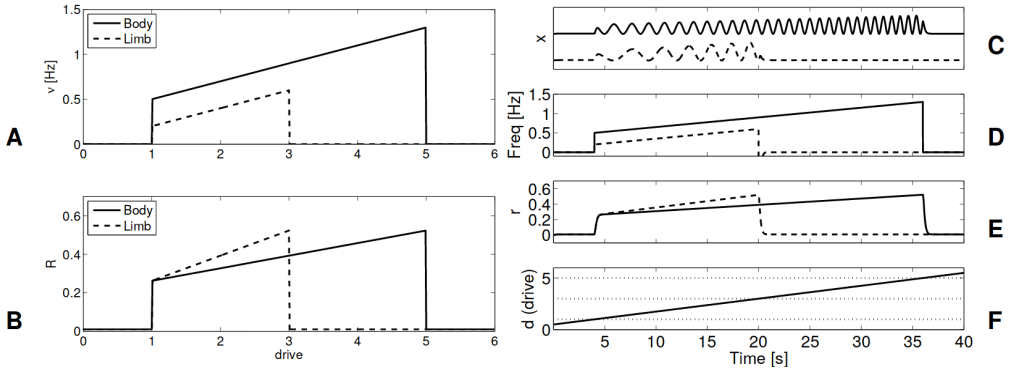
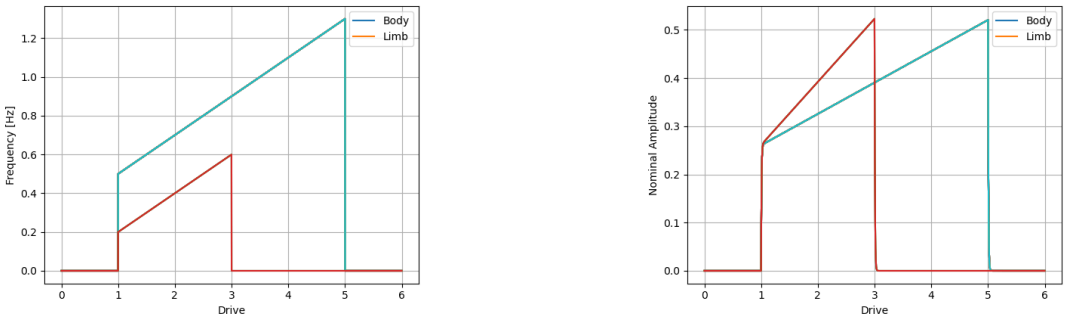


Figure 6: Oscillator properties from [3] supplementary material



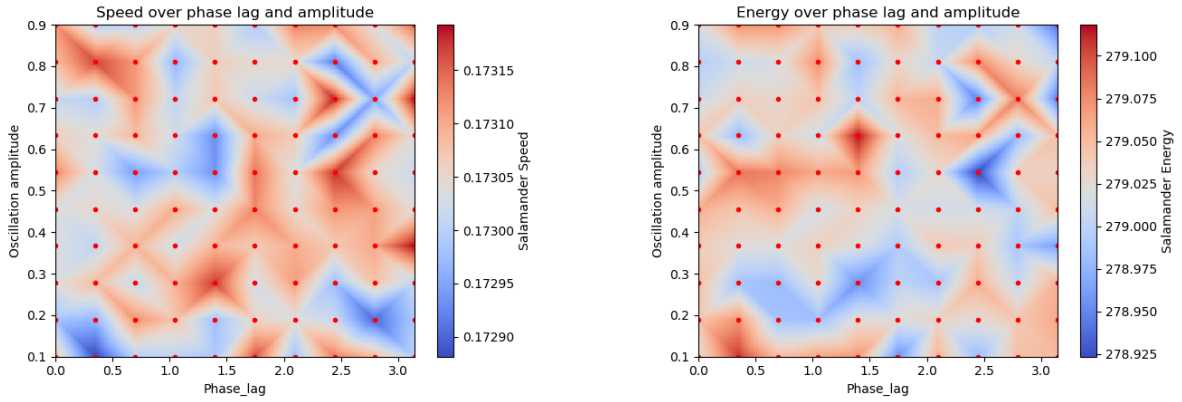
(a) Frequency *(b) Nominal amplitude*

Figure 7: Saturation functions from our network (compare with Fig 6)

8b. Effects of amplitude and phase lags on swimming performance

In this exercise, we study how does phase lag and oscillation amplitude influence the speed and energy. We run a grid search in file exercise_8b.py to explore the robot behavior for different combinations of amplitudes and phase lags. The drive is set to 4 so that the salamander can swim and the frequency is set to 1 Hz. The amplitude will vary from 0.1 to 0.9 in and the phase lag will vary from 0 to π

rads. In total, we use 100 samples (10 different amplitudes and 10 different phase lags). We tried to speed up the simulation by setting `fast` and `headless` to *True*, and it took around 16 minutes to do the simulation (10 secs per simulation).



(a) Speed (m/s) *(b) Energy (J)*

Figure 8: Grid search results

We use the function plot_results.py::exercise8b_gridsearch() to load and plot the logged data from the simulation including 2D/3D plots showing our grid search results. The metrics we are interested in are speed and energy. Speed is computed by adding the mean speed of the salamander head along the x-coordinate and the y-coordinate. Energy is computed by integrating the product of the joint torques and velocities. The grid search results for both metrics are shown in Figure 8. At the end, our aim is to obtain the best combination of amplitude and phase lag which yields the highest speed and the lowest energy. For that purpose, we create another metric called *performance* which is simply the speed divided by the energy. In Figure 9, we observe that the best performance occurs for amplitude=0.55 and phase_lag=2.5 rads.

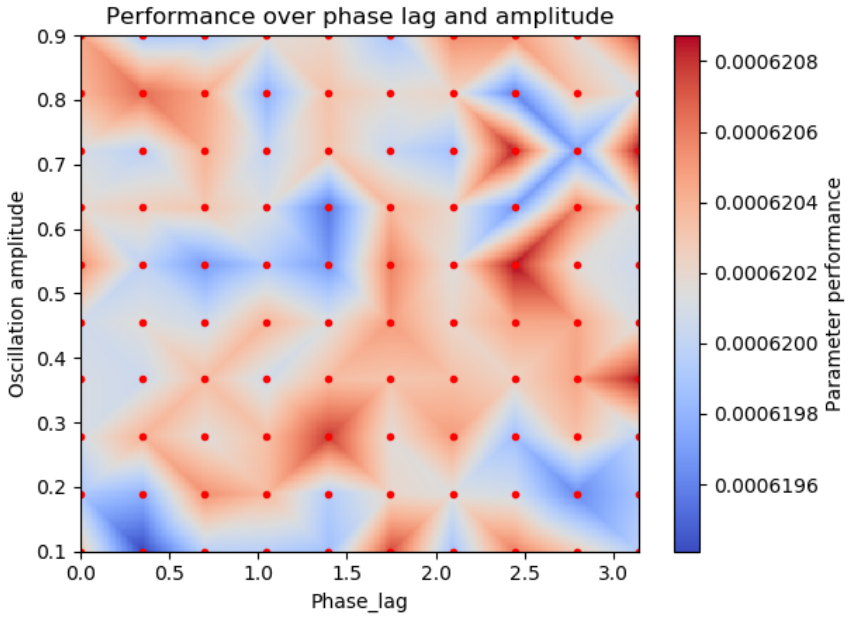
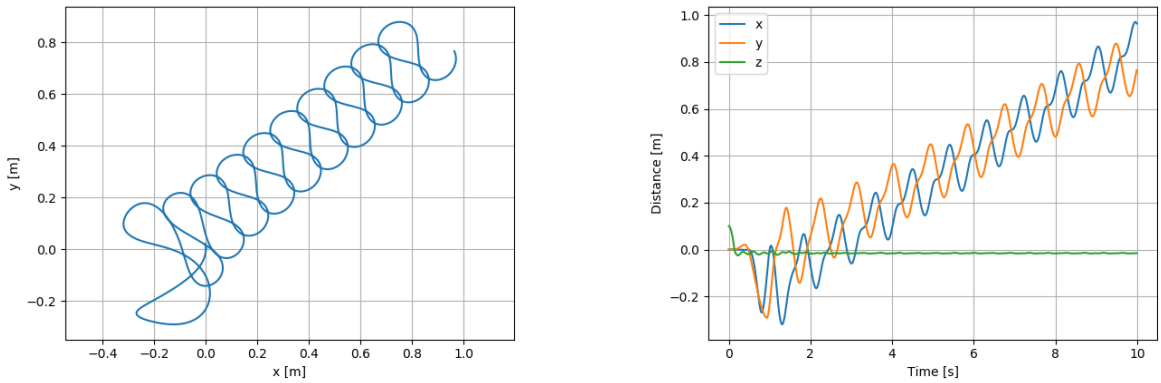


Figure 9: Grid search results for performance

Once we have found the best combination, we plot the salamander trajectory and positions for the

simulation corresponding to the most performing combination of amplitude and phase lag, using the function plot_results.py :: plot_efficient_behaviour(i). The resulting plots are shown in Figure 10 and we observe that, although the salamander is moving forward, there is room for improvement because it is oscillating too much. This issue will be solved in the following exercises.



(a) Trajectory *(b) Positions*

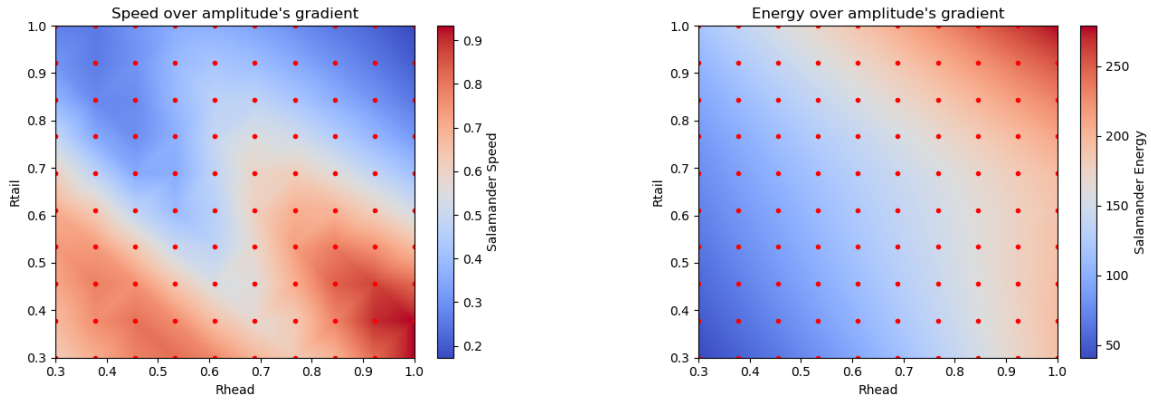
Figure 10: Salamander behaviour for lowest energy and highest speed

8c. Amplitude gradient

In this exercise, we study how does an amplitude's gradient influence the speed and the energy of the salamander. Therefore we used two parameters: amplitudes of the first (Rhead) and last (Rtail) oscillator in the spine (corresponding to the first and last motor). To do so, you can add a parameter amplitudes=[Rhead, Rtail] in simulation_parameters.py::SimulationParameters. The goal of this step is to have a smooth movements in the salamander. When we have the parameters (Rhead) and (Rtail), we just have to compute the gradient by using the numpy function linspace like :

```
np.linspace(Rhead, Rtail, nb_body_joint)
```

Then, we use the function plot_results.py::exercise_8c_plot_gridsearch() to load and plot the logged data from the simulation including 2D/3D plots showing our grid search results. We're interested by the metrics speed and energy. Those two metrics are computed exactly as in Section 8b. The amplitude gradient will vary from 0.3 to 1 for Rhead and Rtail which gives use 100 samples. As before, we tried to speed up the simulation and it took around 16 minutes to compute all the simulations (10 secondes per simulation).



(a) Speed (m/s) *(b) Energy (J)*

Figure 11: Grid search results

As we can see in Figure 11, the speed and the energy of the robot varies as a function of the gradient of the amplitude. First, with the Figure 11a, we observe that if the amplitude of the head is higher that the amplitude of the tail, the robot will swim faster. Second, with Figure 11b, if the amplitude is set to the maximum for the head and the tail, the energy will be maximal which corresponds to the definition of the energy (the more you have strength the more the energy will becomes).

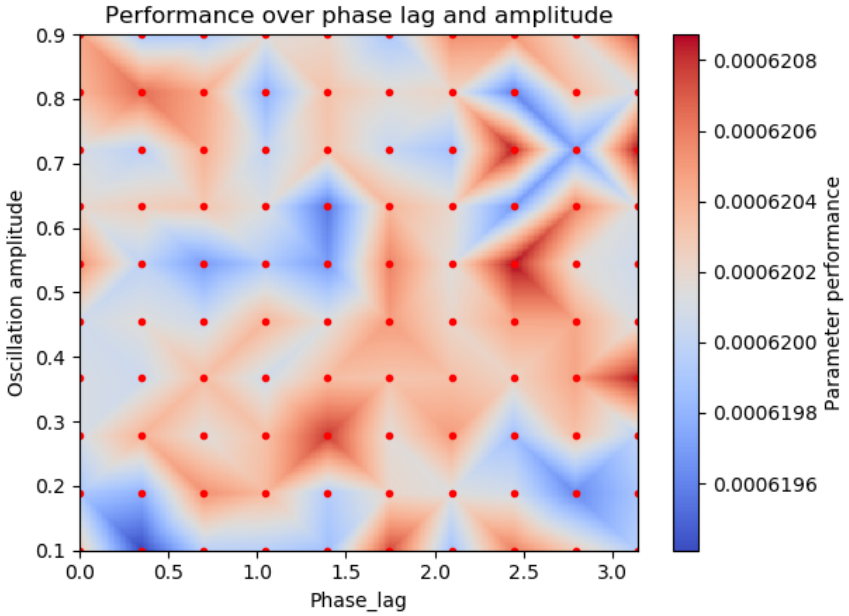
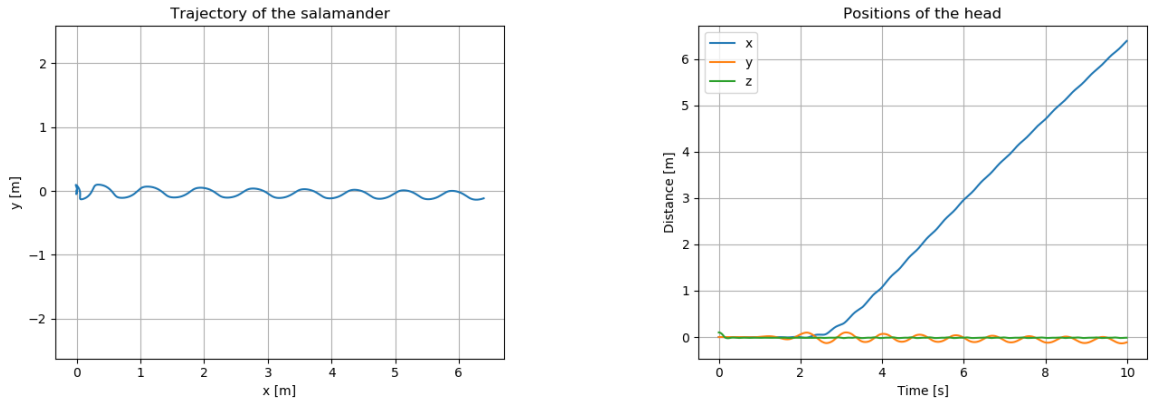


Figure 12: Phase lags along the spine during walking

Finally, as for section 8b, we compute the performance to find the best parameters and as shown in figure 11, the best parameters are Rhead= 0.3 and Rtail = 0.3. In comparison with the exercise_8b.py, the salamander move in a more straight forward direction as shown in the figures below:



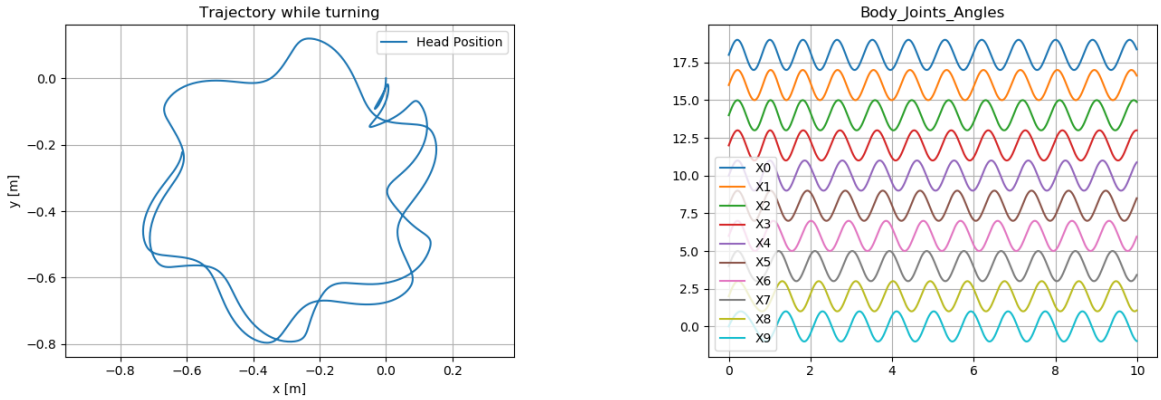
(a) Trajectory *(b) Positions*

Figure 13: Salamander behaviour for lowest energy and highest speed

As we can see in figure 13b where the movement along the x-axis is linear, meaning that its speed is constant. It shows that the body deformation is constant because of the phase lag being constant through the whole robot.

8d. Turning and backwards swimming

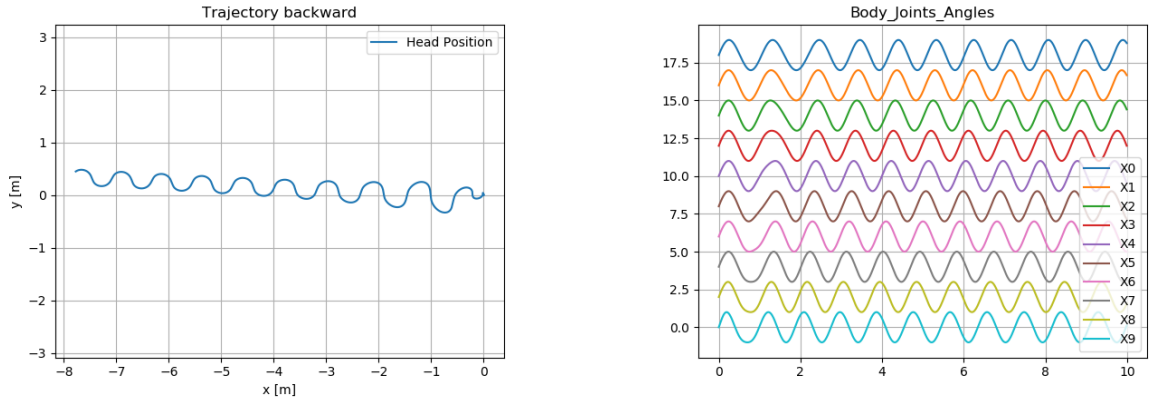
For inducing turning, we can modulate the CPG network by decrease the nominal amplitude of one part of the body. In other words, if we decrease the nominal amplitude of the front part of the body, the salamander will turn left and if we decrease the nominal amplitude of the tail, the salamander will turn right. The figures 14 below show the GPS trajectory and the spine angles while turning.



(a) Trajectory *(b) Spine angles*

Figure 14: Salamander trajectory and spine angles while turning

Now, if we want to let the robot swim backwards, we just have to multiply each phase bias by -1. If we do so, the tail behaviour looks like the head and vice-versa. The figures 15 below show the GPS trajectory and the spine angles while moving backwards.



(a) Trajectory *(b) Spine angles*

Figure 15: Salamander trajectory and spine angles while turning

It's interesting to see that if we let the robot swim backwards, the phase lag comes from the tail to the head.

8f. Limb – Spine coordination

In this part we explore the importance of a proper coordination between the spine and the limb movement for walking which is shown by the walking speed of the salamander. We do a parameter search over the nominal radius and phase offset between limb and body parts in exercise_8f.py to find what are the most optimal values.

Firstly, by changing only the drive to have a walking movement we find that indeed the graph of phase lags along the spine over time looks exactly like a walking movement where the upper part of the body has a single phase lag with the lower part, compared to swimming one which has a constant offset through the whole body.

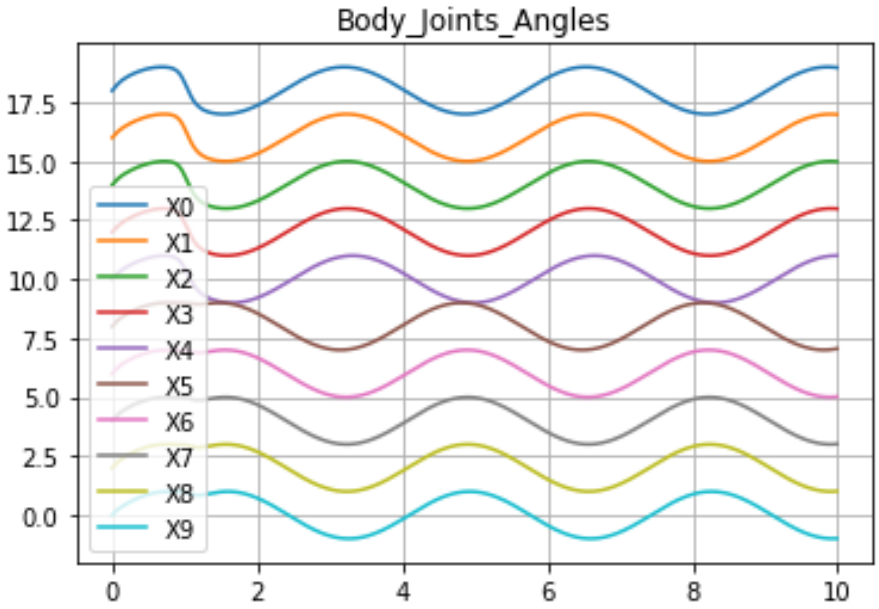
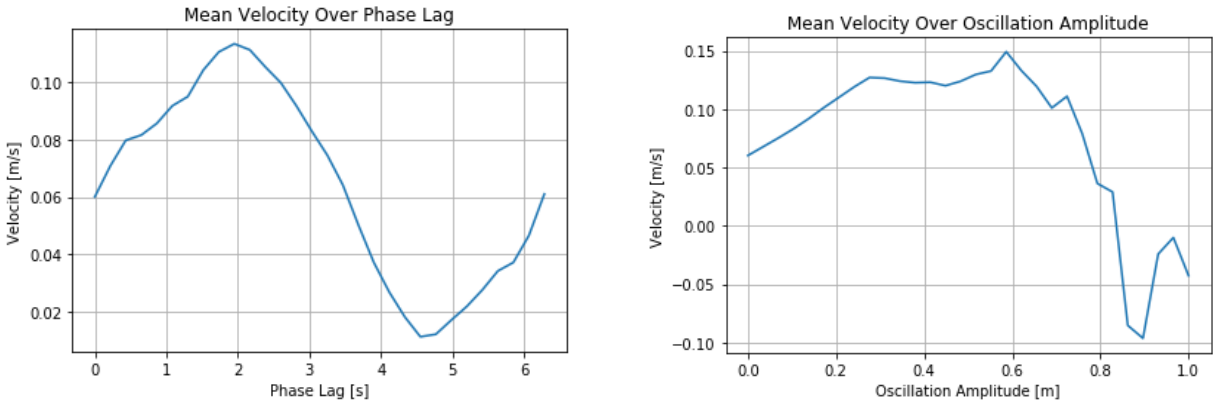


Figure 16: joint angles along the spine, showing the phase lags through the whole body

Then we lock the nominal radius to 0.3 and change the phase offset from 0.1 to 1.0 and plot the speed of the head along the x-axis to compute the optimal value and once computes we do the opposite : Lock the phase offset as the previous found optimal value and do a parameter search over the nominal radius.



(a) Speed over phase offset with nominal radius = 0.3[m] *(b) Speed over oscillation with phase offset = 1.95[s]*

Figure 17: Velocity over phase offset and oscillation amplitude

We can see from the first graph that it has a sinusoidal shape with the highest being at 1.9499540608488373. This plot shows that a lower value is more optimal, which could be explained by how the body moves while walking : The upper and low part of the body undulating in an opposite, symmetrical way. So, if the offset is too big then it will take more time to have a push from the limbs thus be slower overall.

Then the second plot shows that it is pretty constant until around 2π where it drops suddenly, because

the body will turn too much over itself as shown in Figure 18. Otherwise the velocity slowly increases until 0.5862068965517241, the maximum value. This shows that a greater amplitude is better until it is not realistic anymore, because the limbs are able to push the body in a more efficient way.

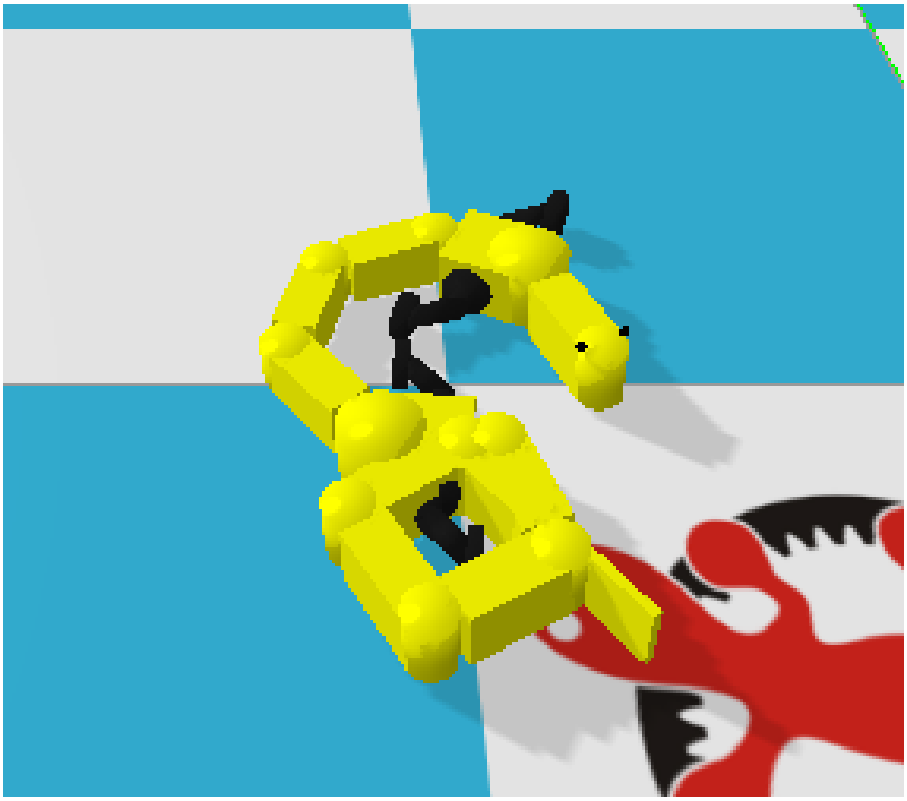
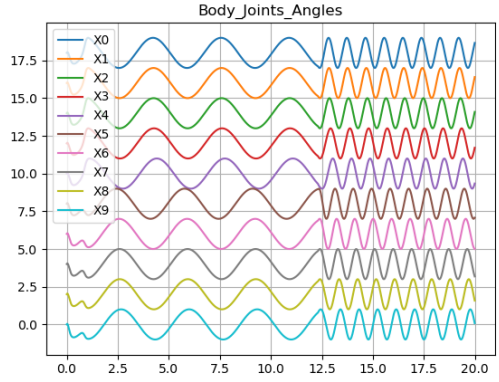


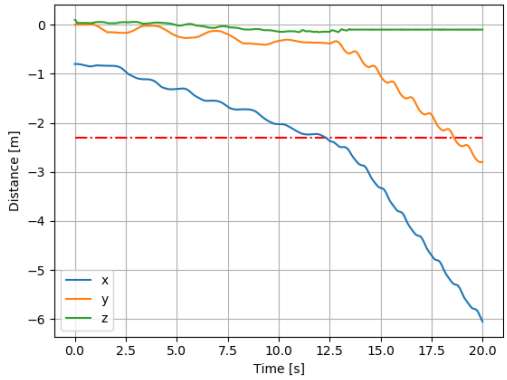
Figure 18: Problem occurring if the oscillation amplitude is too big (here 0.9[m])

8g. Land-to-water transitions

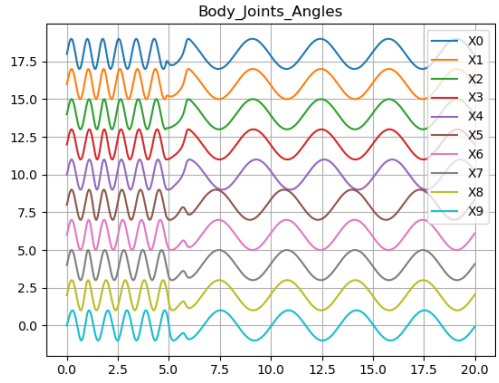
In this final part we will compute the transition from land to water (and water to land), by changing the drive of the salamander depending on its x position. In our code, a drive between 1 and 3 will result to walk, while a drive between 3 and 5 will result to swim. Values in between will slightly change the pace. We used 4 for water and 1.5 for land.



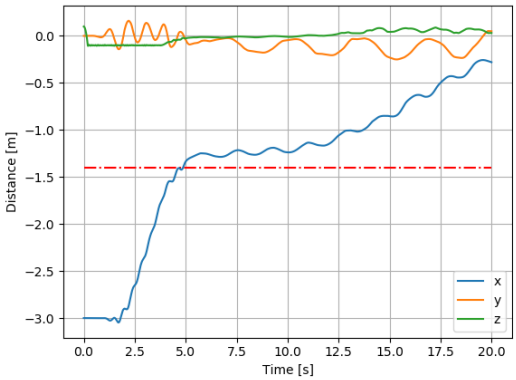
(a) Joint angles for walk to swim transition



(b) Positions for walk to swim



(c) Joint angles for swim to walk transition



(d) Positions for walk to swim

Figure 19: Velocity over phase offset and oscillation amplitude

In those graphs it shows really well the transitions, with the red dotted line showing the coordinate in x where it does the transition. After experimenting with different values, we put the transition from land to water at $-2.3[m]$ and from water to land at $-1.4[m]$. The former lets the whole body of the salamander go into the water before beginning to swim while the latter is deep into land so that the movement of the tail helps going up the slope before beginning to walk.

In the land to water transition we can see that after the transition the salamander goes sideways because he didn't finish a whole cycle of walking before transitioning. In the water to land we can see a slight struggle for one cycle or 2, because of the slope.

The video (that can be found in \Lab8\Report\simulation_final.mp4) shows the whole process of the lab :

1. Walk on land
2. Transition to water
3. Swim in water
4. Turn right (when the salamander's head position in x is over 3.5)
5. Transition to land

References

[1] A. Crespi, K. Karakasiliotis, A. Guignard, and A. J. Ijspeert, "Salamandra robotica ii: An amphibious robot to study salamander-like swimming and walking gaits," *IEEE Transactions on Robotics*, vol. 29, pp. 308–320, April 2013.

[2] K. Karakasiliotis, N. Schilling, J.-M. Cabelguen, and A. J. Ijspeert, "Where are we in understanding salamander locomotion: biological and robotic perspectives on kinematics," *Biological Cybernetics*, vol. 107, pp. 529–544, Oct 2013.

[3] A. J. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen, "From swimming to walking with a salamander robot driven by a spinal cord model," *science*, vol. 315, no. 5817, pp. 1416–1420, 2007.