Marie Anselmet
Inès De Riedmatten
Héloïse Monnet

Student names: Marie Anselmet, Inès De Riedmatten, Héloïse Monnet

*Instructions: Update this file (or recreate a similar one, e.g. in Word) to prepare your answers to the questions. Feel free to add text, equations and figures as needed. Hand-written notes, e.g. for the development of equations, can also be included e.g. as pictures (from your cell phone or from a scanner). **This lab is graded.** and needs to be submitted before the **Deadline : Wednesday 27-05-2020 23:59. You only need to submit one final report for all of the following exercises combined henceforth.** Please submit both the source file (*.doc/*.tex) and a pdf of your document, as well as all the used and updated Python functions in a single zipped file called final_report_name1_name2_name3.zip where name# are the team member's last names. Please submit only one report per team!*

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).

NOTE : The session this week will be an introduction to the final project. You will be installing the PyBullet physics engine will and get to familiarise with it. You will start implementing the CPG network which will eventually control the robot.

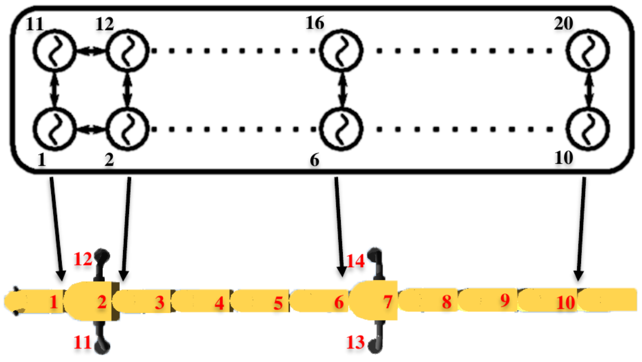


Figure 1: A double chain of oscillators controlling the robot's spine.

Code organization

- **exercise_all.py** - A convenient file for running all exercises. Note you can also run the simulations in parallel by activating `parallel=True`. *You do not need to modify this file.*
- **network.py** - This file contains the different classes and functions for the CPG network and the Ordinary Differential Equations (ODEs). You can implement the network parameters and the ODEs here. Note that some parameters can be obtained from pythonController.py to help you control the values.
- **robot_parameters.py** - This file contains the different classes and functions for the parameters of the robot, including the CPG network parameters. You can implement the network parameters here. Note that some parameters can be obtained from SimulationParameters class in simulation_parameters.py and sent by exercise_#.py to help you control the values (refer to example).

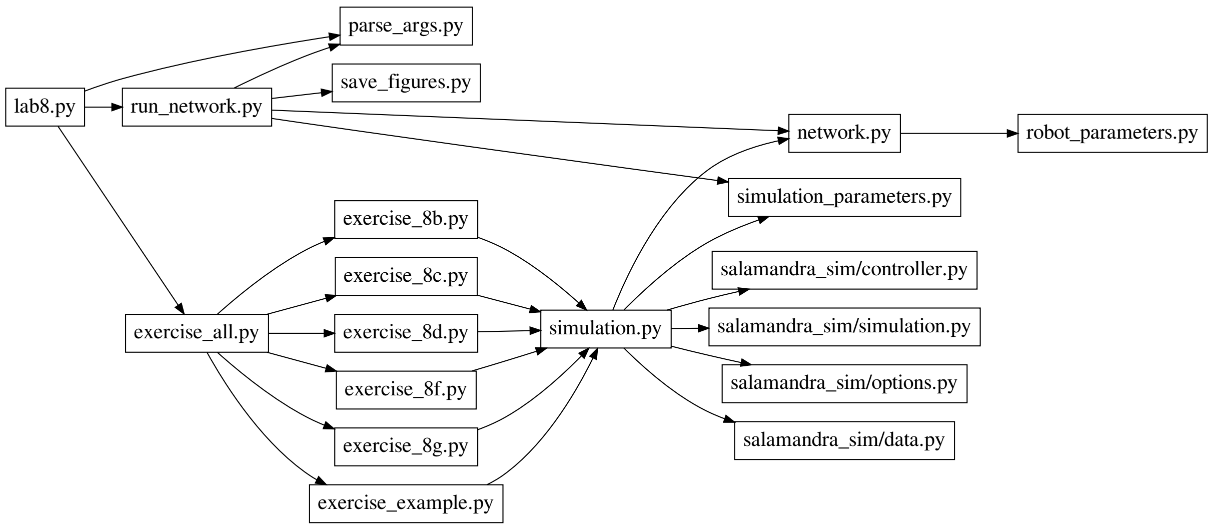


Figure 2: Exercise files dependencies. In this lab, you will be modifying `run_network.py`, `network.py`, `robot_parameters.py` and `simulation_parameters.py`

- **simulation__parameters.py** - This file contains the SimulationParameters class and is provided for convenience to send parameters to the setup of the network parameters in robot__parameters.py. All the values provided in SimulationParameters are actually logged in cmc__robot.py, so you can also reload these parameters when analyzing the results of a simulation.
- **run__network.py** - By running the script from Python, PyBullet will be bypassed and you will run the network without a physics simulation. Make sure to use this file for question 8a to help you with setting up the CPG network equations and parameters and to analyze its behavior. This is useful for debugging purposes and rapid controller development since running the Pybullet simulation takes more time.
- **parse__args.py** - Used to parse command line arguments for run__network.py and plot__results.py and determine if plots should be shown or saved directly. *You do not need to modify this file.*
- **save__figures.py** - Contains the functions to automatically detect and save figures. *You do not need to modify this file.*
- **test__sr2.py** - This is a file to verify that Pybullet was installed correctly. It is important to make sure that this file works as it will be necessary for the project. *You do not need to modify this file.*
- **exercise__example.py** - Contains the example code structure to help you familiarize with the other exercises. *You do not need to modify this file.*
- **exercise__#.py** - To be used to implement and answer the respective exercise questions. Note that exercise__example.py is provided as an example to show how to run a parameter sweep. Note that network parameters can be provided here.
- **exercise__all.py** - A convenient file for running different exercises depending on arguments. See **lab8.py** for an example on how to call it. *You do not need to modify this file.*
- **simulation.py** A simulation function is provided for convenience to easily run simulations with different parameters. You are free to implement other functions to run simulations as necessary.
- **plot__results.py** - Use this file to load and plot the results from the simulation. This code runs with the original pythonController provided.

Prerequisites

To have all the necessary python packages necessary to complete the final project, do the following.

Clone the latest version of the exercise repository. Navigate to the location of your repository in the terminal and execute the following,

```
>> pip install -r requirements.txt
```

Running the simulation

You can run a simulation example with **exercise_example.py**. You should see the Salamandra robotica model floating on the water. At this point you can now start to work on implementing your exercises.

Marie Anselmet
Inès De Riedmatten
Héloïse Monnet

Questions

The exercises are organized such that you will have to first implement the oscillator network model in run_network.py code and analyze it before connecting it to the body in the physics simulation. Exercise 8a describes the questions needed to implement the oscillator models. After completing exercise 8a you should have an oscillator network including both the spinal CPG and limb CPG. Using the network implemented in exercise 8a you can explore the swimming, walking and transition behaviors in the Salamandra robotica II model using the pybullet simulation and complete exercises 8b to 8g.

8a. Implement a double chain of oscillators along with limb CPGs

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. Implement the double chain oscillator model using the functions `network.py::network_ode`. Test your implementation by running the network using `run_network.py`. For the network parameters check lecture slides (pay attention to different number of segments). You can also find more information in [3] (especially in the supplementary material). You can set all the network parameters in the `robot_parameters.py::RobotParameters`. To facilitate your work, you could start by only implementing the network for the body oscillators ($i = [0, ..., 19]$) and ignoring the leg oscillators ($i = [20, ..., 23]$). Refer to network::RobotState and robot_parameters.py::-RobotParameters for the dimensions of the state and the network parameters respectively.

2. Implement the output of your CPG network to generate the spinal joint angles according to equation 3. Implement this in the function `network.py::motor_output`. Verify your implementation in by running the Python file `run_network.py`.

3. Implement a drive and show that your network can generate swimming and walking patterns similarly to [3]. Try to reproduce the plots in 3 and 4

   Hint: The state for the network ODE is of size 48 where the first 24 elements correspond to the oscillator phases θ_i of the oscillators and the last 24 elements correspond to the amplitude r_i. The initial state is set in the init of network.py::SalamanderNetwork.

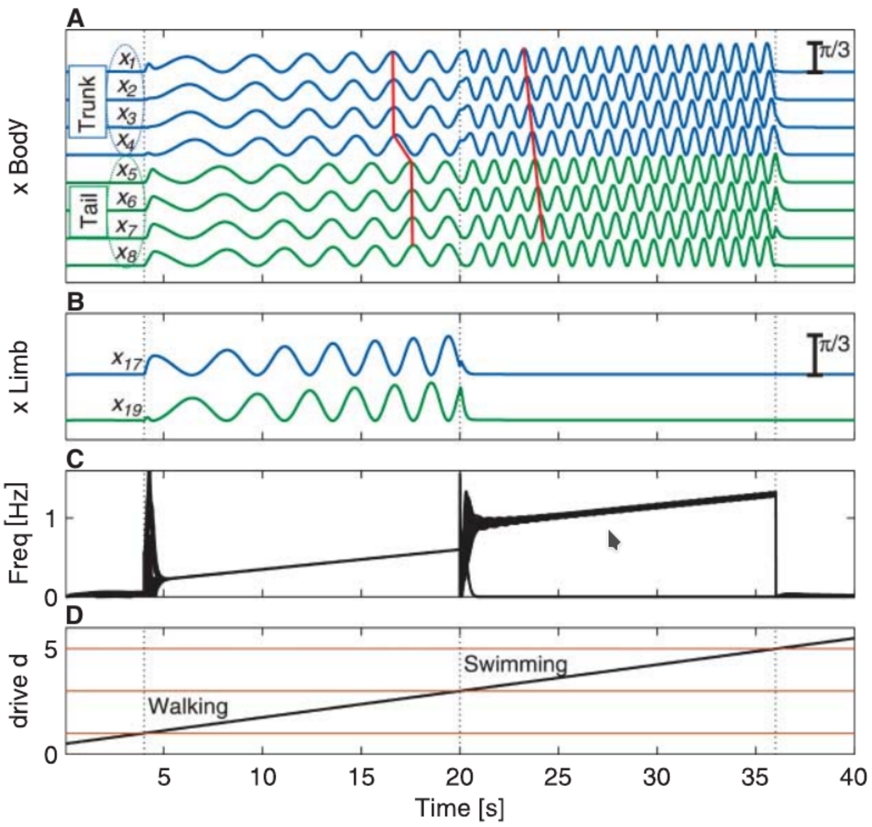


Figure 3: Oscillator patterns from [3], see [3] for details

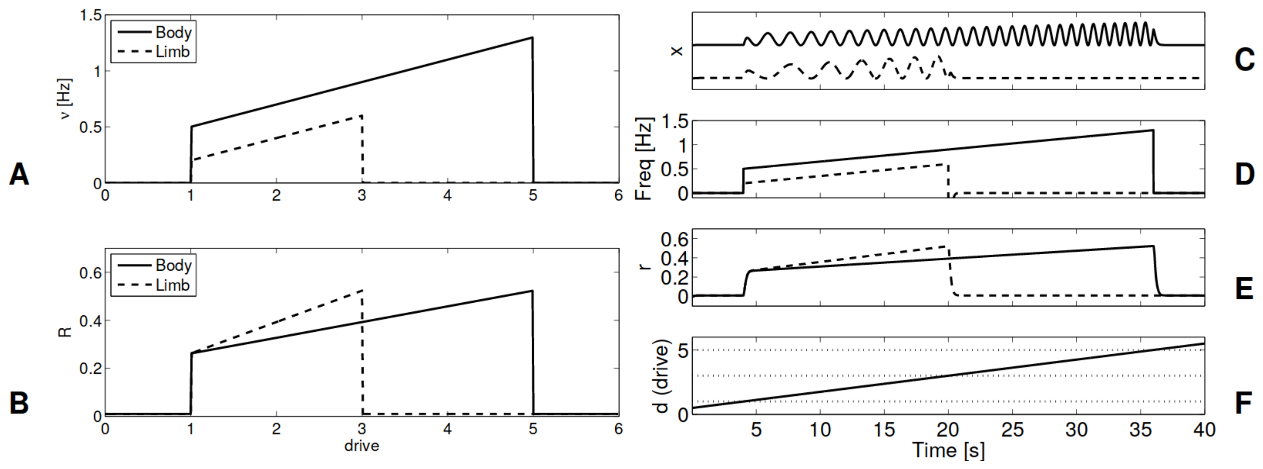
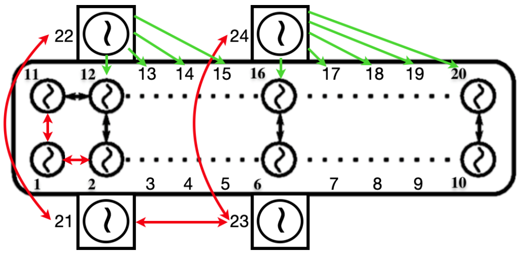


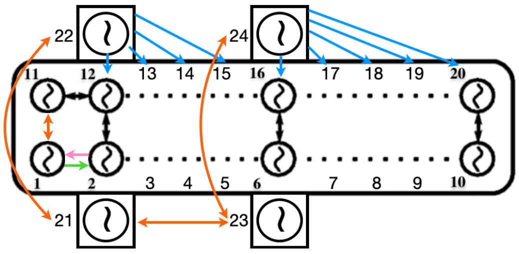
Figure 4: Oscillator properties from [3] supplementary material, see [3] for details

Marie Anselmet
Inès De Riedmatten
Héloïse Monnet

Salamander model and weights and phase biases matrices used for the implementation

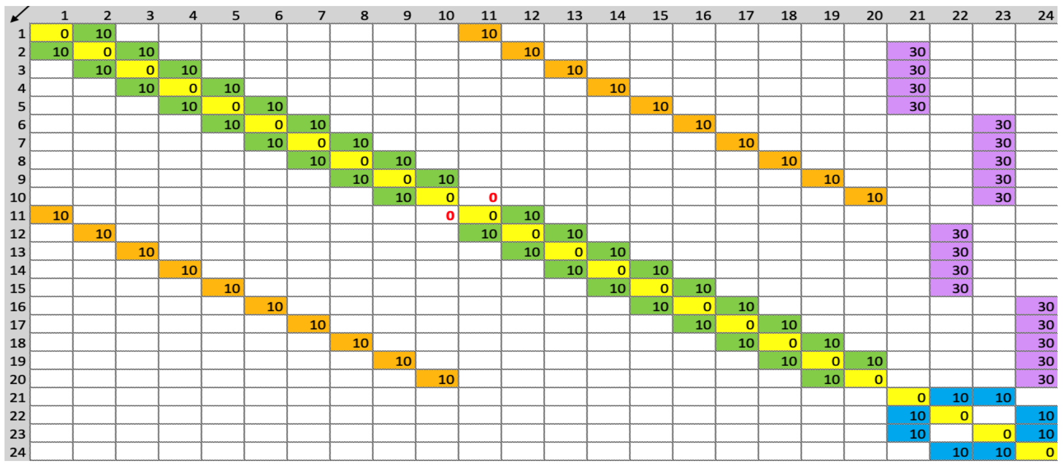


(a) *Weights w_{ij} (from j to i). Red arrows : w_{ij} = 10, green arrows : w_{ij} = 30*



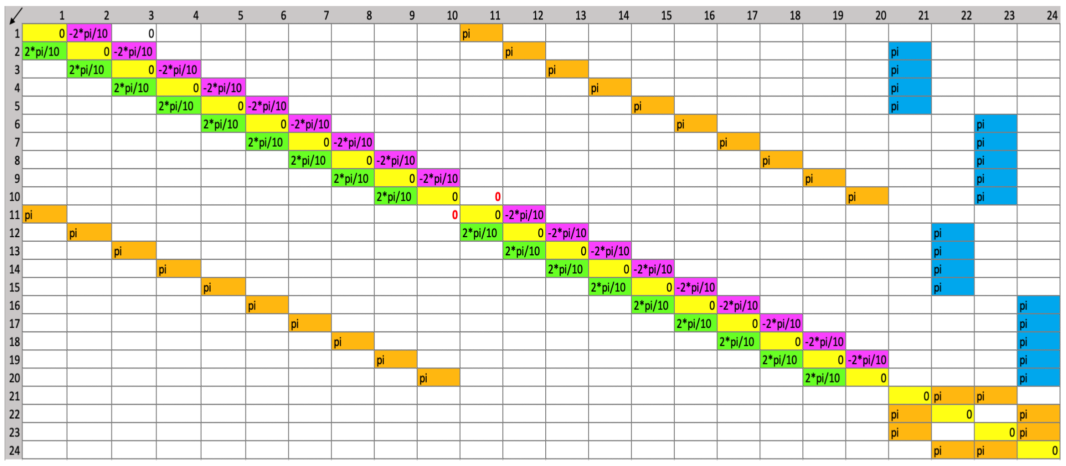
(b) *Phase biases ϕ_{ij} (from j to i). Green arrows : ϕ_{ij} = $2\pi/10$, pink arrows : ϕ_{ij} = -$2\pi/10$, orange and blue arrows : ϕ_{ij} = π*

Figure 5: Weights and phase biases in the Salamander model.



↙	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	10									10													
2	10	0	10									10									30			
3		10	0	10									10								30			
4			10	0	10									10							30			
5				10	0	10									10						30			
6					10	0	10									10							30	
7						10	0	10									10						30	
8							10	0	10									10					30	
9								10	0	10									10				30	
10									10	0	0									10			30	
11	10									0	0	10												
12		10									10	0	10									30		
13			10									10	0	10								30		
14				10									10	0	10							30		
15					10									10	0	10						30		
16						10									10	0	10							30
17							10									10	0	10						30
18								10									10	0	10					30
19									10									10	0	10				30
20										10									10	0				30
21																					0	10	10	
22																					10	0		10
23																					10		0	10
24																						10	10	0

Figure 6: Table of weights w_{ij} (from j to i). Green: downwards/upwards in body CPG, orange: contralateral in body CPG, violet: from limb to body CPG, blue: within the limb CPG.



↙	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	-2*pi/10	0								pi													
2	2*pi/10	0	-2*pi/10									pi									pi			
3		2*pi/10	0	-2*pi/10									pi								pi			
4			2*pi/10	0	-2*pi/10									pi							pi			
5				2*pi/10	0	-2*pi/10									pi						pi			
6					2*pi/10	0	-2*pi/10									pi							pi	
7						2*pi/10	0	-2*pi/10									pi						pi	
8							2*pi/10	0	-2*pi/10									pi					pi	
9								2*pi/10	0	-2*pi/10									pi				pi	
10									2*pi/10	0	0									pi			pi	
11	pi									0	0	-2*pi/10												
12		pi									2*pi/10	0	-2*pi/10									pi		
13			pi									2*pi/10	0	-2*pi/10								pi		
14				pi									2*pi/10	0	-2*pi/10							pi		
15					pi									2*pi/10	0	-2*pi/10						pi		
16						pi									2*pi/10	0	-2*pi/10							pi
17							pi									2*pi/10	0	-2*pi/10						pi
18								pi									2*pi/10	0	-2*pi/10					pi
19									pi									2*pi/10	0	-2*pi/10				pi
20										pi									2*pi/10	0				pi
21																					0	pi	pi	
22																					pi	0		pi
23																					pi		0	pi
24																						pi	pi	0

Figure 7: Table of phase biases ϕ_{ij} (from j to i). Green: upwards in body CPG, pink: downwards in body CPG, orange: contralateral in body CPG and within the limb CPG, blue: from limb to body CPG (phase offset). Read from columns to rows.

Swimming and walking patterns generated by the network implemented

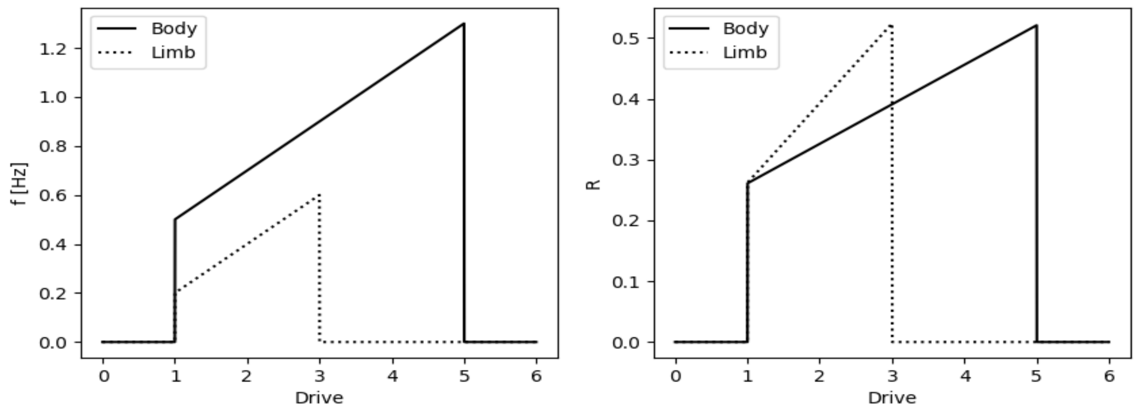


Figure 8: Saturation functions providing the intrinsic frequency f (Left) and amplitude R (Right) for body and limb oscillators.

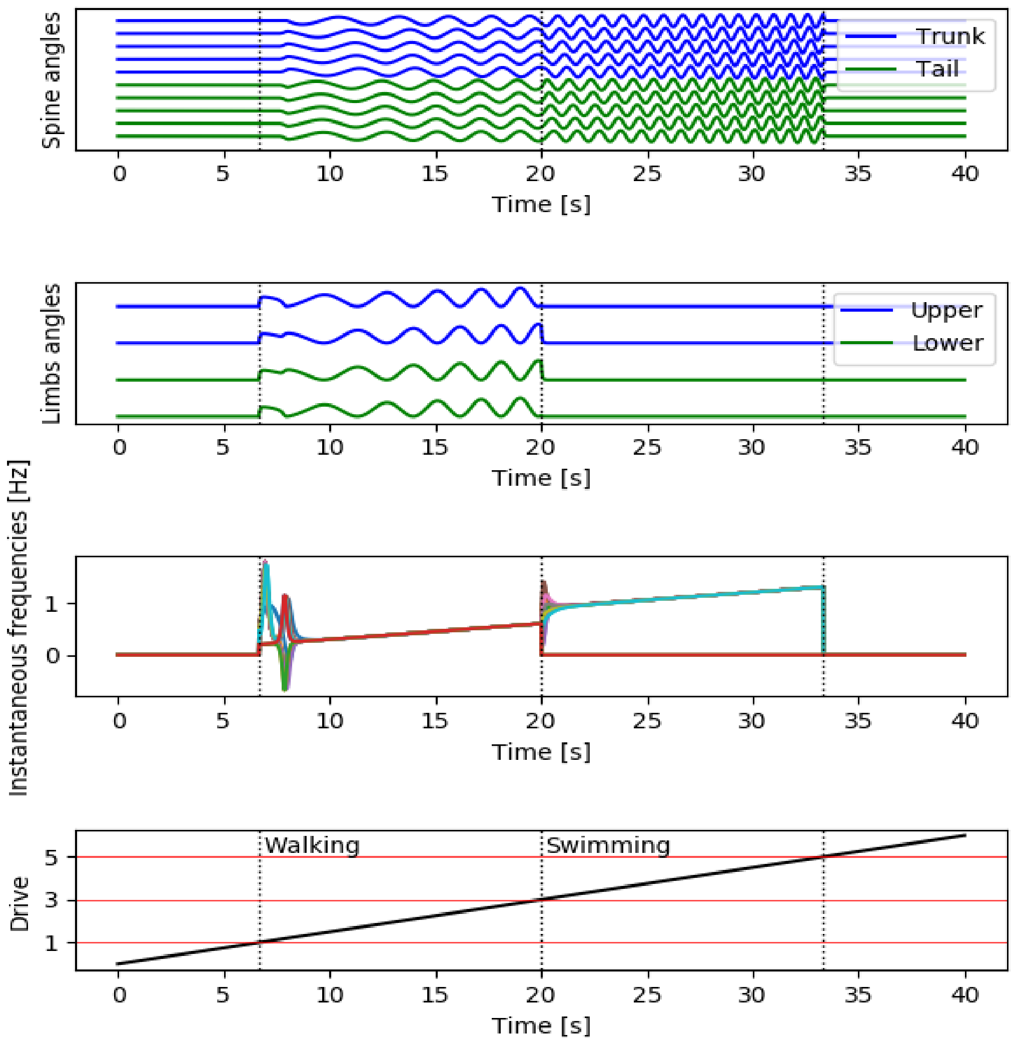


Figure 9: Switching from walking to swimming: activity of the CPG model when the drive is progressively increased. (1) Spine angles. (2) Limbs angles. (3) Instantaneous frequencies. (4) Linear increase of the drive d applied to all oscillators. The horizontal red lines correspond to the lower and upper oscillation thresholds for limb and body oscillators.

Figure 8 shows the saturation functions determining the intrinsic frequency f and amplitude R for the body and limbs oscillators as a function of the drive d. For the body oscillators, these functions are defined as:

$$f = 0.2d + 0.3 \qquad \text{for } 1 < \text{d} < 5 \text{ (0 otherwise)}$$
$$R = 0.065d + 0.196 \qquad \text{for } 1 < \text{d} < 5 \text{ (0 otherwise)}$$

Whereas for the limbs oscillators, they are defined as:

$$f = 0.2d \qquad \text{for } 1 < \text{d} < 3 \text{ (0 otherwise)}$$
$$R = 0.131d + 0.131 \qquad \text{for } 1 < \text{d} < 3 \text{ (0 otherwise)}$$

As stated in hypothesis 3 and 4 of the paper [3], the limbs cannot oscillate at high frequencies and have lower intrinsic frequencies than the body, for the same drive.

Figure 9 shows the activity of the resulting CPG model when switching from walking to swimming. As it can be seen on the fourth plot, the drive is linearly increased from 0 to 6 during the time of the simulation. When it reaches a value of 1 at t=7s, the salamander switches to a walking gait and when it reaches a value of 3 at t=20s, the salamander switches to a swimming gait. After t=33s, when the drive is higher than 5, the salamander stops moving.
The spine angles plot illustrates the transition from standing waves - with synchrony in the trunk, synchrony in the tail and an antiphase relation between the two - during walking, to traveling waves during swimming.
In the plot showing the limbs angles, it can be seen that the limbs move only during the walking gait. Ipsilateral fore- and hindlimbs are in antiphase. The limbs are computed as constantly rotating, but the limb angles shown in figure 9 are the projection according to the equation $x_i = r_i(1+\cos(\theta_i))$ in [3].
Finally, the instantaneous frequencies are plotted in the third plot. The variations among individual oscillators at times t=7s and t=20s correspond to brief accelerations and decelerations before resynchronization.

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

Now that you have implemented the controller, it is time to run experiments to study its behaviour. How does phase lag and oscillation amplitude influence the speed and energy? Use the provided exercise_8b.py to run a grid search to explore the robot behavior for different combinations of amplitudes and phase lags. Use plot_results.py to load and plot the logged data from the simulation. Include 2D/3D plots showing your grid search results and discuss them. How do your findings compare to the wavelengths observed in the salamander?

- **Hint 1:** To use the grid search, check out the example provided in exercise_example.py. This function takes the desired parameters as a list of SimulationParameters objects (found in simulation_parameters.py) and runs the simulation. Note that the results are logged as simulation_#.h5 in a specified log folder. After the grid search finishes, the simulation will close.
- **Hint 2:** An example how to load and visualise grid search results is already implemented in plot_results.py::main(). Pay attention to the name of the folder and the log files you are loading. Before starting a new grid search, change the name of the logs destination folder where the results will be stored. In case a grid search failed, it may be safer to delete the previous logs to avoid influencing new results by mistake.
- **Hint 3:** Estimate how long it will take to finish the grid search. Our suggestion is to choose wisely lower and upper limits of parameter vectors and choose a reasonable number of samples. To speed-up a simulation, make sure to run the simulation in fast mode and without GUI as shown in exercise_example.py or using `-fast` and `-headless` in the Python command line (Use `-help` for more information).

- **Hint 4:** Energy can be estimated by integrating the product of instantaneous joint velocities and torques. Feel free to propose your own energy metrics, just make sure to include the justification.

First, it is important to recall that for swimming, the phase lag strongly influences the "S-shape" undulation of the traveling wave observed in axial muscles. Indeed, the phase lag is defined as the lag divided by the cycle duration, and as for the lamprey, we expect a wavelength of roughly one body length for the swimming salamander, so a total phase lag of 2π along the spine oscillators. Therefore, the default phase lag between the spine oscillators was defined in our code as $2\pi/10$ for the rest of the questions, and for running this grid search, we looked at values around $2\pi/10$, with a π-shift. Indeed, in this project, the spine of the salamander is modelled with a double chain of 10 oscillators, as shown in figure 1. More precisely, the values tested were the following: $\pi/10$, $2\pi/10$, $3\pi/10$, $2\pi/5$, $\pi/2$.

On the other hand, the nominal radius R_i is defined as the inverse of the curvature of the circle that best describes the trajectory path of the salamander [1]. Therefore, the larger the nominal radius (or nominal amplitude) is, the larger the curvature of the salamander undulation will be. Indeed, according to equation 2, the isolated oscillator amplitude converges towards its nominal amplitude, and since all R_i are set to the same value here, all amplitudes converge towards this same R_i value. However, note that we are not interested in the convergence factor a since it only determines how quickly the amplitude variables r_i converge towards R_i, and the 30s of simulation are sufficient to reach convergence with the default value of a=24. For the nominal amplitude, the values tested in the grid search were close to the range of relevant values regarding the saturation function providing the nominal amplitude (see B on figure 4). More precisely, we tested values spaced by 0.1 in the range [0.2, 0.6].

The results of this grid search are shown in the 2D heatmap plots of figure 10.

In order to assess the "quality" of locomotion resulting of each combination of phase lag and nominal amplitude, several criteria were considered. First, the forward speed and the distance traveled by the salamander were determined. In addition, the energy needed to produce the observed displacement was computed in order to provide an estimate of the efficiency of the salamander locomotion. However, it is important to notice here that since the simulation initialization relies on many random parameters, it takes some time for the salamander to reach a stable and efficient locomotion. For this reason the speed, the distance traveled and the energy were computed only after 20s of simulation. A pseudo Cost of Transport function was also defined in this perspective of evaluating the locomotion's efficiency. More precisely, the pseudo Cost of Transport function criterion was defined here as the ratio of the distance traveled by the salamander over the resulting energy consumption. To be consistent with unities, we should have taken into account the mass of the salamander as well as the gravitational constant g, but it was decided to omit this factor since the latter would not have changed our optimization problem consisting in maximizing the inverse of this pseudo Cost of Transport function. Thus, our COT^{-1} has the units of $s^2/(kg*m)$.

It can be seen in figure 10 that the larger and faster forward displacements are obtained for both small phase lags and small nominal amplitudes values. Nonetheless, these parameters are not optimal in terms of energy consumption. Indeed, if we only consider the energy consumption, it seems that large nominal amplitudes are less advantageous. Finally, by looking at the inverse of the Cost of Transport criterion defined above, it can be seen that a maximum seems to be reached for a phase lag of $\pi/2$ and a nominal amplitude of 0.2.

By replacing them into the context, these estimated optimal parameters correspond to a total lag of 5π along the spine, meaning that the observed wavelength is smaller than one body length. Nonetheless, if this does not correspond with what is normally observed in living salamanders, we could assimilate this undulation pattern to a more "eel-shaped" swimming pattern, possibly due to some simplifications of the simulation set-up. For the nominal amplitude, it seems that large undulations enable a faster locomotion, but in such configurations, a large part of the required energy is dissipated and consumed in drag for example.

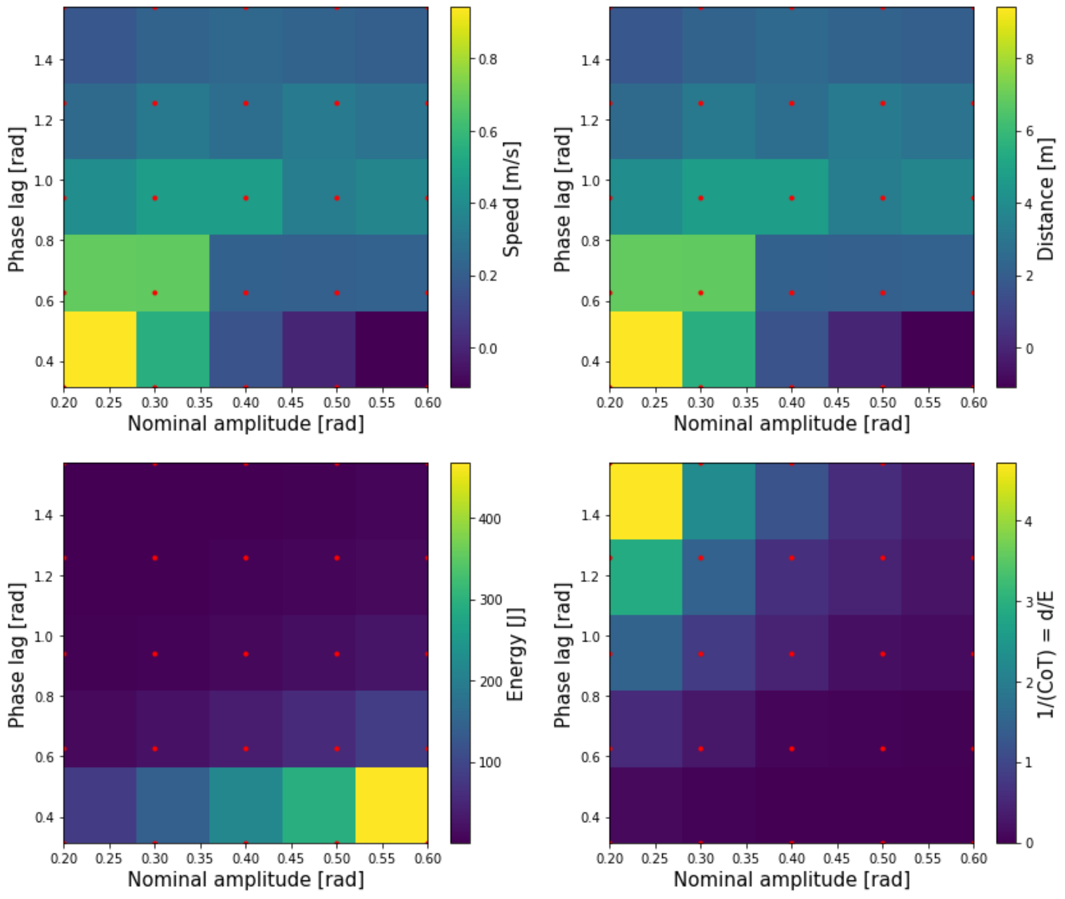


Figure 10: Grid search for phase lags of $\pi/10$, $2\pi/10$, $3\pi/10$, $2\pi/5$, $\pi/2$, and nominal amplitudes in [0.2, 0.6] by steps of 0.1, during swimming. Drive = 4, frequency = 1.1Hz. Simulation lasts 30 seconds. The energy, the speed, the distance traveled as well as the pseudo Cost of Transport function (CoT) were computed after 20s of simulation.

8c. Amplitude gradient

1. So far we considered constant undulation amplitudes along the body for swimming. Implement a linear distribution of amplitudes along the spine, parametrized with 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. Don't forget to modify robot_parameters.py::RobotParameters::set_nominal_amplitudes() and interpolate the amplitude gradient between values Rhead and Rtail within the function. Note that you can then provide this amplitudes parameter from exercise_8c.py.

2. Run a grid search over different values of parameters Rhead and Rtail (use the same range for both parameters). How does the amplitude gradient influence swimming performance (speed, energy)? Include 3D plots showing your grid search results. Do it once, for frequency 1Hz and total phase lag of 2π along the spine.

3. How is the salamander moving (with respect to different body amplitudes)? How do your findings in 2) compare to body deformations in the salamander? Based on your explorations, what could be possible explanations why the salamander moves the way it does?

We implemented an amplitude gradient (=[Rhead,Rtail]) between the head and the tail of the salamander in robot_parameters.py::RobotParameters::set_nominal_amplitudes(). The linear distribution of the amplitudes along the spine was then multiplied by the nominal amplitudes. The goal was to obtain a smaller amplitude at the head and a bigger amplitude at the tail of the salamander. Indeed, this characteristic improves the quality of swimming of the animal and results in an optimal S-shape. As before, we checked several parameters: the energy consumed, the speed and the distance traveled. These parameters were calculated after 20 seconds of simulation, as the simulation required time to begin. Overall, the simulations lasted 30 seconds.

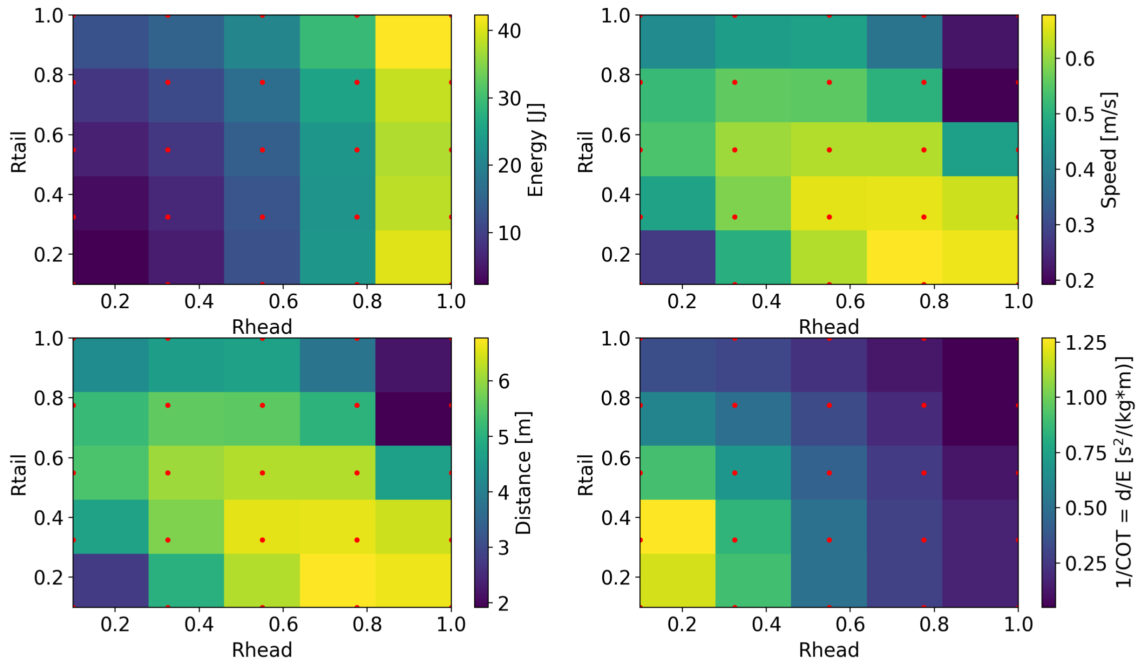
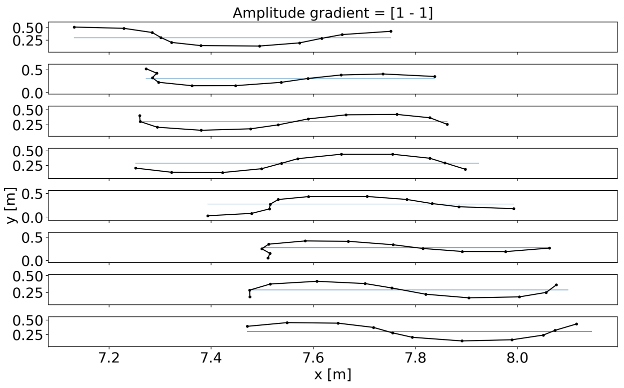


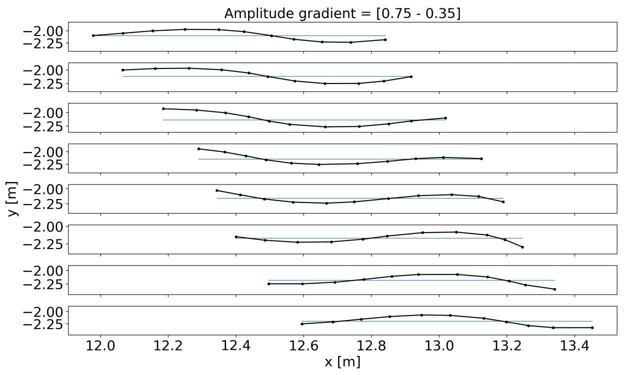
Figure 11: Grid search for amplitude gradient between 0.1 and 1, during swimming. Amplitude gradient = [Rhead, Rtail], drive = 4, frequency = 1Hz, total phase lag = 2π. Simulation lasted 30 seconds. The energy, the speed and the distance traveled were computed after 20s of simulation.

We summarized this analysis by computing the reverse of the "pseudo" Cost Of Transport (COT) = d/E (see figure 11), as explained in question 8b. Indeed, as we want to minimize the energy consumed E and to maximize the distance traveled d, we searched the maximum of d/E = COT^{-1}. We also tested the interaction of the speed and the energy with a grid search looking for the maximum of v/E, but it led to similar result than with COT^{-1}, due to the similarity between distance travelled and speed.

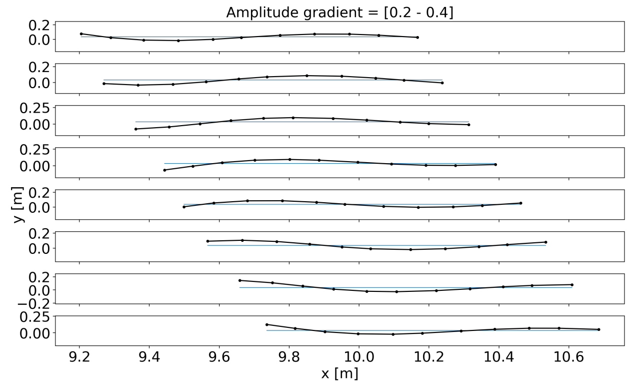
On figure 11, the grid search for the energy shows that the bigger the Rhead, the higher the energy. This means that having a high head amplitude consumes a lot of energy for the salamander to swim. If the Rtail is small, the S-shape of the salamander is reversed (see figure 12b): the curvature of the head is bigger than the curvature of the tail. During the simulation, it seems that only the head is leading the swimming. This configuration requires a lot of energy. On the contrary, if Rhead is small and Rtail is big (see figure 12d), the bending of the body is reversed. We can see in figure 11 that a too big Rtail consumes a lot of energy too, and does not lead to higher speed or distance traveled. Thus, this configuration is not optimal. If both Rtail and Rhead are big, the salamander has an exaggerated S-shape (see figure 12a) that leads to the highest energy consumed in figure 11 and very small speed and distance traveled. Overall, we remark that it costs less to have bigger amplitude at the tail than at the head and smaller Rhead and Rtail consumes less energy. Concerning the distance traveled and the speed, we can see that both grid searches lead to similar results, which is expected as the salamander swims in a straight line. Interestingly, the salamander swims faster and further when it has a higher Rhead and a smaller Rtail, but it requires more energy. On the COT^{-1} grid search of figure 11, we clearly see a maximum around a gradient of amplitude of [0.2, 0.4]. This is the most optimized gradient of amplitude in our analyses. Even though it does not lead to the highest speed, it has the advantage of not requiring high energy.



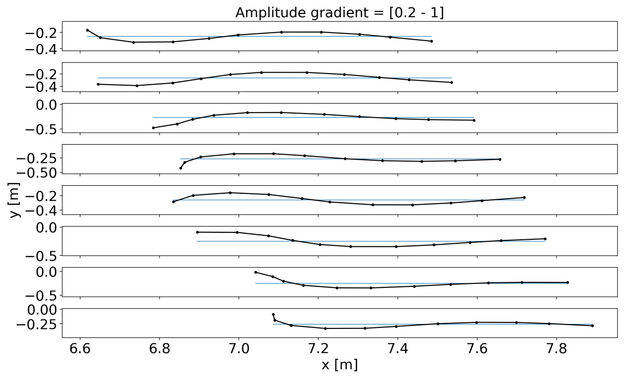
(a) No amplitude gradient, highest energy



(b) Highest speed/distance traveled



(c) Maximized COT^{-1}

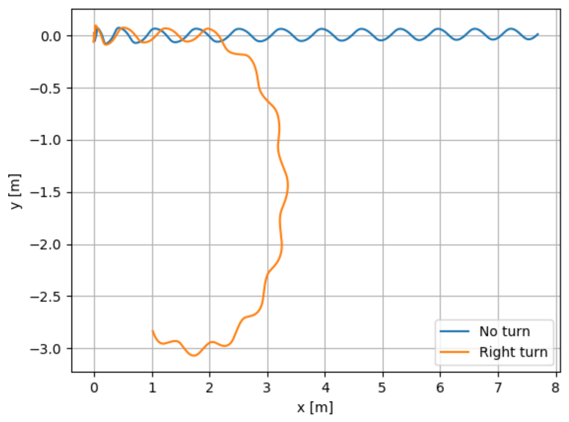


(d) Small Rhead, big Rtail

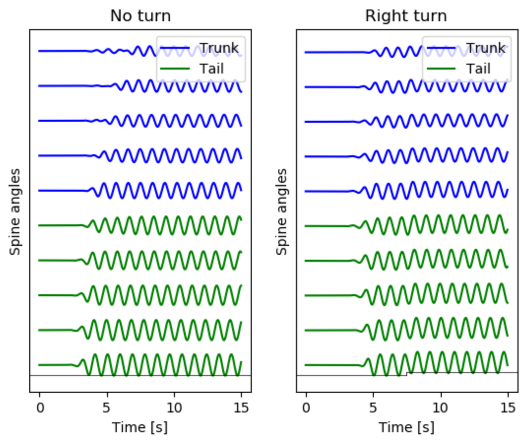
Figure 12: (a)-(d): 8 successive midline profiles showing the x-y behavior of the salamander, during swimming. First profile at the top, last profile at the bottom of each subplot. Time between each profile = 133ms, drive = 4, intrinsic frequency = 1Hz, phase lag = 2π.

8d. Turning and backwards swimming

1. How do you need to modulate the CPG network (network.py) in order to induce turning? Implement this in the model and plot example GPS trajectories and spine angles.



(a) Trajectory



(b) Spine angles

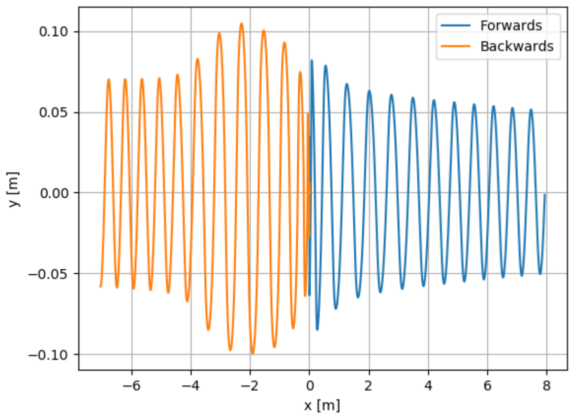
Figure 13: Comparison of the trajectory and of the spine angles when the salamander turns or not. Simulation parameters used: duration = 15s, amplitude gradient = [0.2, 0.4], drive = 4. Right turning induced at 7.5s such that turn = $[f_{left}, f_{right}]$ = [1.2, 0.8].

As it is described in the supplementary material of [3], when the difference between the left and right drive is not too large, the oscillators remain synchronized despite the differences in intrinsic frequencies. The asymmetry results only in a change in oscillation amplitudes with the side receiving higher drive oscillating at higher amplitudes. This leads to an offset in the set points sent to the joint motor controllers, and the robot turns towards the side receiving the highest drive.

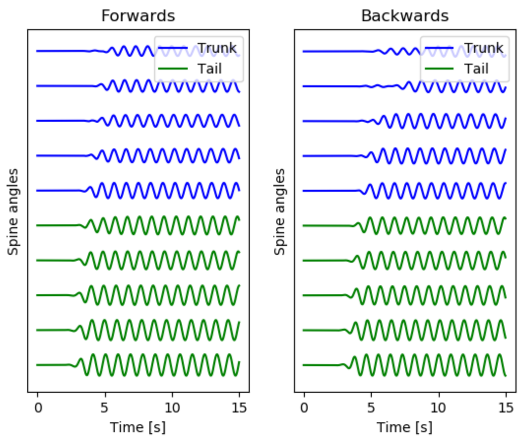
Thus in order to induce turning, we decided to multiply the nominal amplitudes R (just after their definition by the saturation function) of the left and of the right body oscillators by a different factor f. More precisely, a parameter turn=$[f_{left}, f_{right}]$ was added such that f_{left} should be greater than f_{right} in order for the salamander to turn to the right.

Figure 13a shows the trajectory of the salamander during two simulations: one in which it swims straight and the other in which it turns to the right (from 7.5s after the beginning of the simulation). In figure 13b, the salamander spine angles are compared for the two simulations. In the second simulation, in which the salamander begins to turn to the right at t=7.5s, we can clearly notice a change of offset at that time, as expected.

2. How could you let the robot swim backwards? Explain and plot example GPS trajectories and spine angles.



(a) Trajectory



(b) Spine angles

Figure 14: Comparison of the trajectory and of the spine angles when the salamander swims forwards or backwards. Simulation parameters used: duration = 15s, amplitude gradient = [0.2, 0.4], drive = 4, phase lag for forward swimming = $2\pi/10$ *and phase lag for backward swimming =* $-2\pi/10$.

As explained in lecture 6, changing the intrinsic frequency of some oscillators leads to changes in the phase lags and that could be a mechanism for backward swimming. Thus it was decided to reverse the phase lag from $2\pi/10$ to $-2\pi/10$, i.e. the phase biases ϕ_{ij} downwards in body CPG were set to $-2\pi/10$ whereas phase biases ϕ_{ij} upwards in body CPG were set to $2\pi/10$. This should reverse the traveling wave and induce the salamander to swim backwards.

Figure 14a shows the trajectory of the salamander when it swims forwards and backwards. In figure 14b, the salamander spine angles are plotted and look the same for the two simulations. This is due to the cosine in equation 3. Indeed, as cosine is an even function (i.e. $cos(x) = cos(-x)$), the reversal of the phase lag is not visible in the spine angles plot.

8e. Cancelled

8f. Limb – Spine coordination

In this next part we will explore the importance of a proper coordination between the spine and the limb movement for walking.

1. Change the drive to a value used for walking and verify that the robot walks.

With a drive of 2.0, a walking gait was clearly observed. So for this question, when not precised, the drive was set to 2.0.

2. Analyze the spine movement: What are your phase lags along the spine during walking? How does the spine movement compare to the one used for swimming?

In this question, we focus on the spine movement. For both walking and swimming experiments, the simulation lasted 20s, and the drive was set to 2.0 for walking and 4.0 for swimming.
In the figures 15 and 16 below, corresponding to the walking and swimming gaits respectively, the spine angles versus the time of the simulation were first plotted.
The vertical phase differences (here we just considered downwards coupling) between the spine oscillators were plotted as a function of the time of the stimulation. The trunk oscillators and the tail oscillators were labeled in blue and green respectively, and the last oscillator belonging to the trunk as well as the first oscillator belonging to the tail for each side of the spine were labeled in red.
Finally, the stable phase differences, characterizing the couplings described above, were estimated over the last 2s of both simulations. The x-axis of these sub-plots indicate the coupling index as follows: the coupling between oscillators 1 and 2 is indexed as the coupling number 1, and so on.

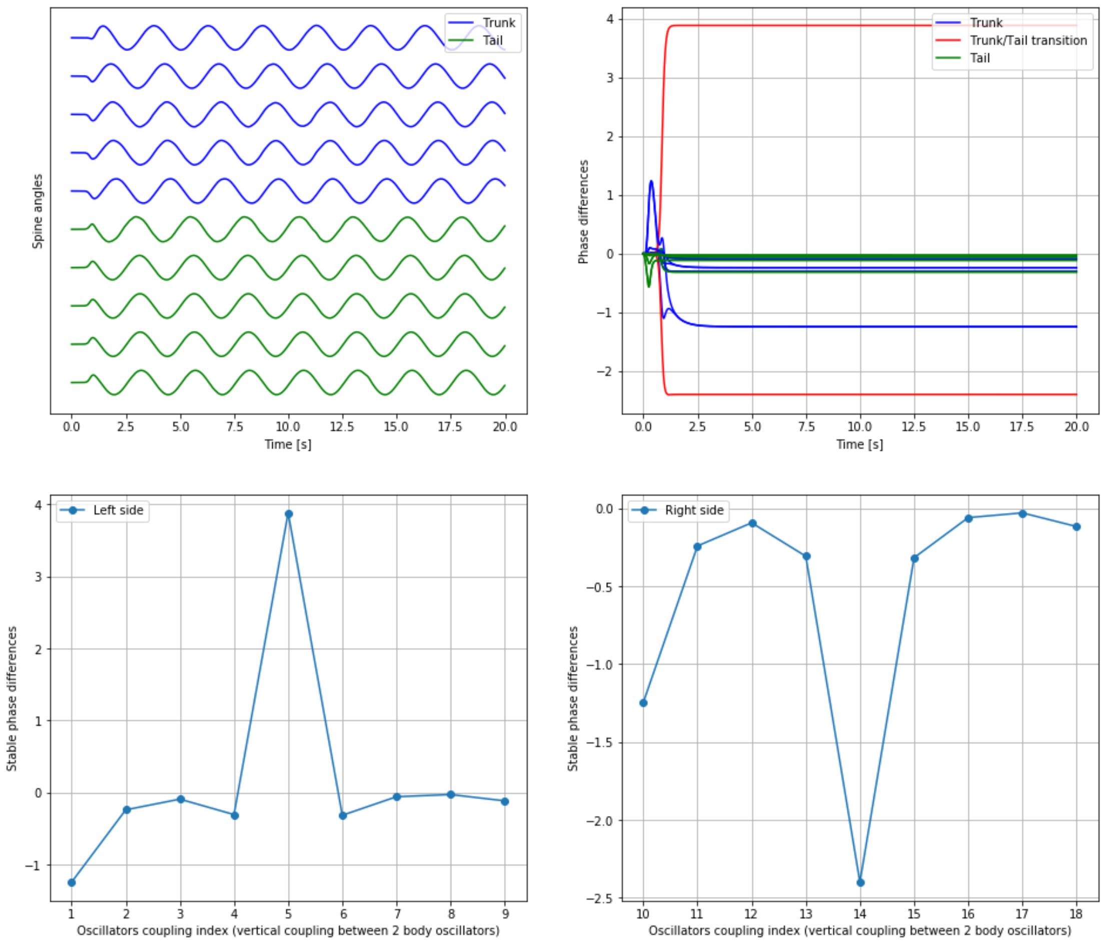


Figure 15: Phase lags along the spine during walking. Here the downwards couplings along the spine are considered. Drive = 2.0. The phases are in radians.

As said before, it can be seen on these plots that the frequency of the oscillations observed in the spine angles are larger for swimming than for walking, and this is due to the definitions of the different saturation functions.
Concerning the phase differences, it can be seen that for both swimming and walking gaits, the convergence towards a synchronization state seems to be reached quite rapidly, leading to a phase-locked system. However, if the phase lags seem to reach almost constant values between neighbors for swimming (close to a lamprey-like swimming), they seem to reach substantially different constants values for walking. Indeed, it can be seen on figures 15 and 16 that the phase differences are not constant along the spine for both gaits, and that the differences are higher for walking.
Concerning swimming, each side of the spine obtains the same bell-shaped profile for the stable phase differences estimated during the 2 last seconds of the simulation, as seen in lecture 6. It has yet to be noticed that in lecture 6, there was no notion of side and a simple chain of vertically coupled oscillators was implemented, so we were not able to see the nice symmetry obtained here between contralateral oscillators.
Finally, for the walking gait, a peak corresponding to the transition between the trunk and the tail of the salamander can be observed. Moreover, the left and right side oscillators of the spine seem to exhibit opposite phase locking steady state, which can be explained by the fact that during walking, both ipsilateral and contralateral limbs are in antiphase, with the larger curvature observed at the level of the trunk-tail transition, thus explaining the observed peaks.

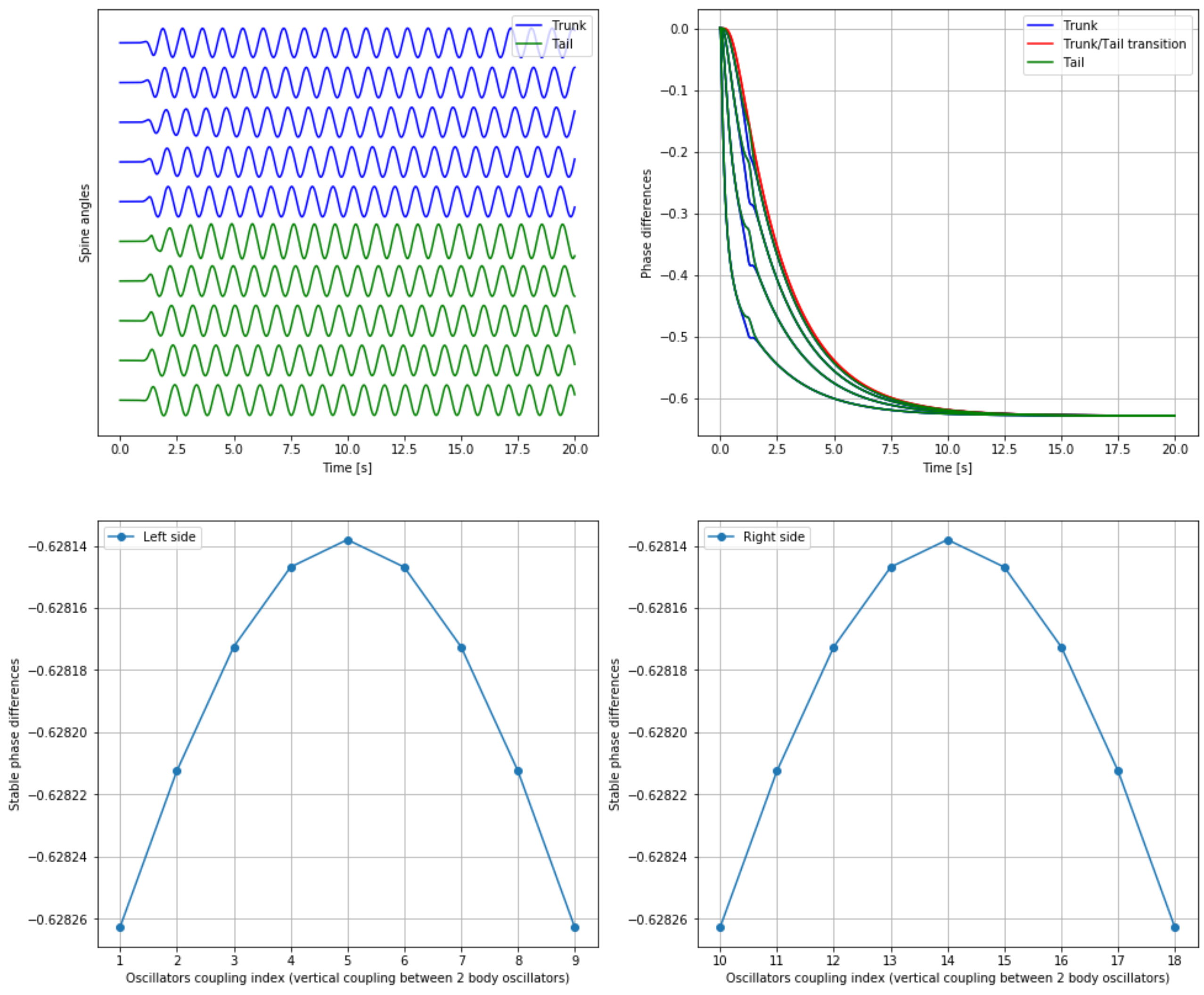


Figure 16: Phase lags along the spine during swimming. Here the vertical and downwards couplings are considered. Drive = 4.0. The phases are in radians.

3. Notice that the phase between limb and spine oscillators affects the robot's walking speed. Run a parameter search on the phase offset between limbs and spine. Set the nominal radius R to 0.3 [rad]. Include plots showing how the phase offset influences walking speed and comment the results. How do your findings compare to body deformations in the salamander while walking?

The forward velocity according to the phase offset from the limbs to the spine oscillators was plotted in figure 17 below. The nominal radius was set to 0.3, the drive to 2.0 and the rest of the parameters as before. The simulation lasted 30s. For the same reasons as mentioned before (stabilization), the forward speed was estimated after 20s of simulation. Based on what has been done in [1], the values tested for the phase offset were in the range $[-\pi, \pi]$, by steps of $\pi/9$. The exact step size does not matter here, but it was a way to have a sufficient density of points for smoothing as much as possible the plotted curve.

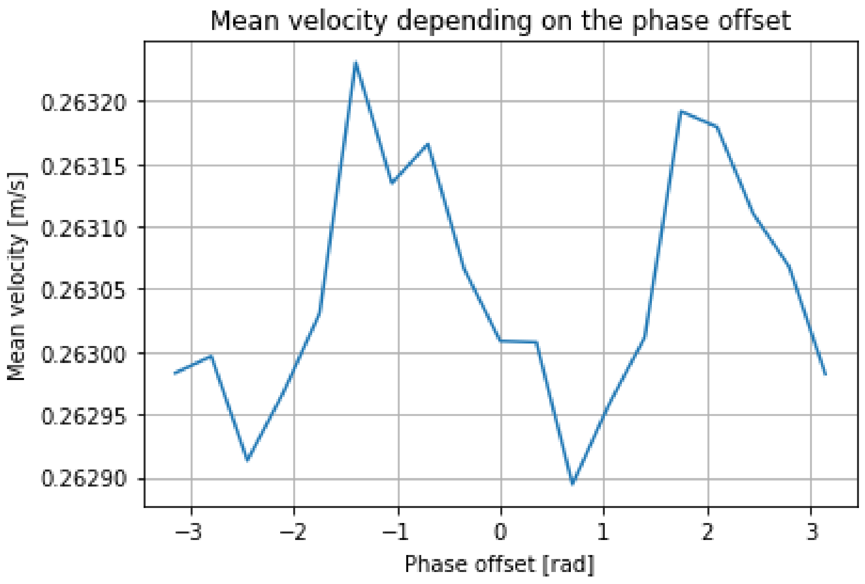


Figure 17: Forward velocity in function of the phase offset between the limbs and the spine. R = 0.3 radians, drive = 2.0, simulation duration = 30s.

By looking at the phase offset between the limbs and the spine, we are interested here on the body-limbs coordination, i.e. on the relationship between the rotation of the limbs and the phase of the body wave. It can be seen on figure 17 that the velocity profile in the tested range of phase offsets is roughly symmetric, with a slight right-shift from 0. Two main local maxima can be observed at rougly $-\pi/2$ and at a slight shift to the right from $\pi/2$. However, values of offsets close to 0 seem to lead to small measured velocities, contrary to what was observed in [1], and this could be explained by the fact that the set-up was not exactly the same, especially the double chain of body oscillators. But the main principle that matters is the following: it seems that the optimal locomotion for walking corresponds to a maximal protraction of a forelimb on one side, while the trunk is maximally bent toward the other side, because this optimizes the stride length [1].

4. Explore the influence of the oscillation amplitude along the body with respect to the walking speed of the robot. Run a parameter search on the nominal radius R with a fixed phase offset between limbs and the spine. For the phase offset take the optimal value from the previous sub-exercise. While exploring R, start from 0 (no body bending).

Include plots showing how the oscillation radius influences walking speed and comment on the results.

The forward velocity according to the nominal radius was plotted in figures 18 and 19 below. The phase offset was set to π, the drive to 2.0 and the rest of the parameters as before. The simulation lasted 30s. For the same reasons as mentioned before (stabilization), the forward speed was still estimated after 20s of simulation. 20 values were tested for the nominal radius in the range [0, 0.5]. Several intrinsic frequencies values were also implemented.

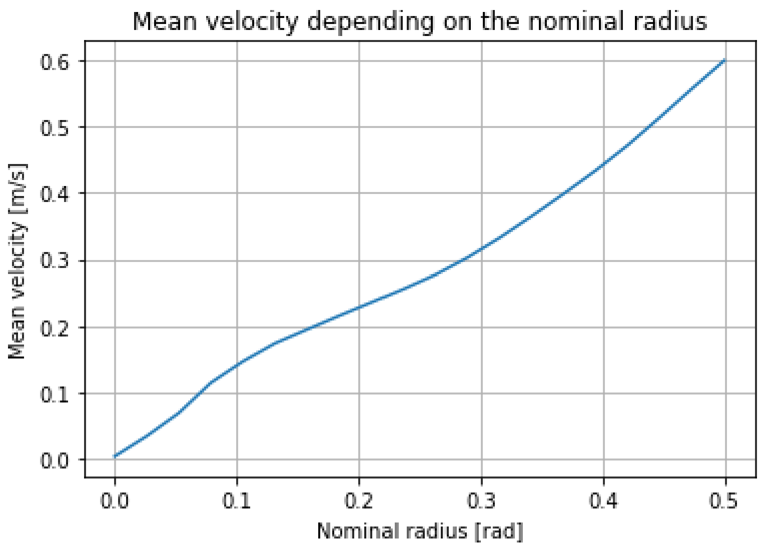


Figure 18: Forward velocity in function of the nominal radius. Phase offset between the limbs and the spine = π rad. Drive = 2.0. Simulation duration = 30s. Intrinsic frequencies = 0.4Hz.

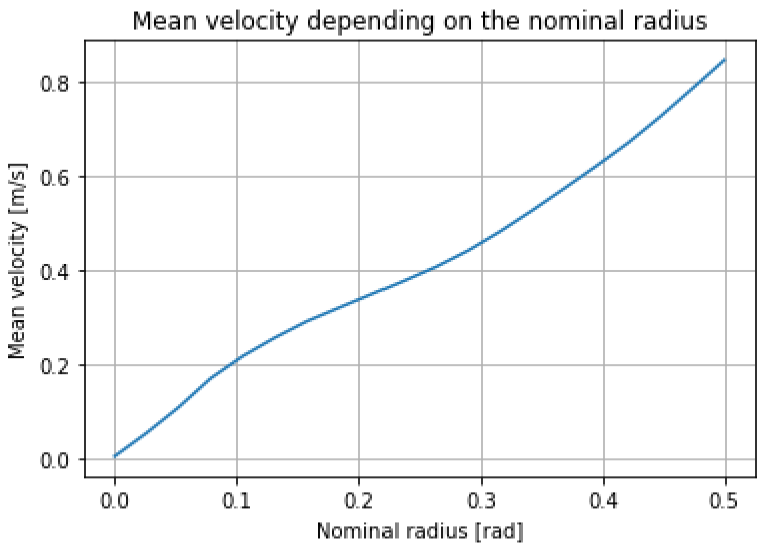


Figure 19: Forward velocity in function of the nominal radius. Phase offset between the limbs and the spine = π rad. Drive = 2.0. Simulation duration = 30s. Intrisic frequencies = 0.6Hz.

It can be seen in figures 18 and 19 that in the observed range of values, an increase in the nominal radius leads to an increase in the measured forward velocity of the salamander. In addition, it can be observed that the speed also increases as the frequency increases. Both observations are coherent, since the curvature of the salamander during walking enables a larger protraction of the limbs, and since a frequency increase speeds up the cycles. Nonetheless, it has to be noticed that real salamanders tend to rely more on the control of the frequency than on the modulation of the nominal radius to vary their speed [1]. Finally, the considerations mentioned in the questions 8b) and 8c) concerning the energy consumption and the efficiency of the locomotion should also be taken into account to evaluate the walking efficiency, even if the stride length characteristic, in some part, already goes in this way.

8g. Land-to-water transitions

1. In this exercise you will explore the gait switching mechanism. The gait switching is generated by a high level drive signal which interacts with the saturation functions that you should have implemented in 8a. Implement a new experiment which uses the x-coordinate of the robot in the world retrieved from the GPS sensor reading (Check simulation.py for an example on how to access the gps data). Based on the GPS reading, you should determine if the robot should walk (it's on land) or swim (it reached water). Depending on the current position of the robot, you should modify the drive such that it switches gait appropriately.
2. Run the Pybullet simulation and report spine and limb angles, together with the x coordinate from the GPS signal. Record a video showing the transition from land to water and submit the video together with this report.

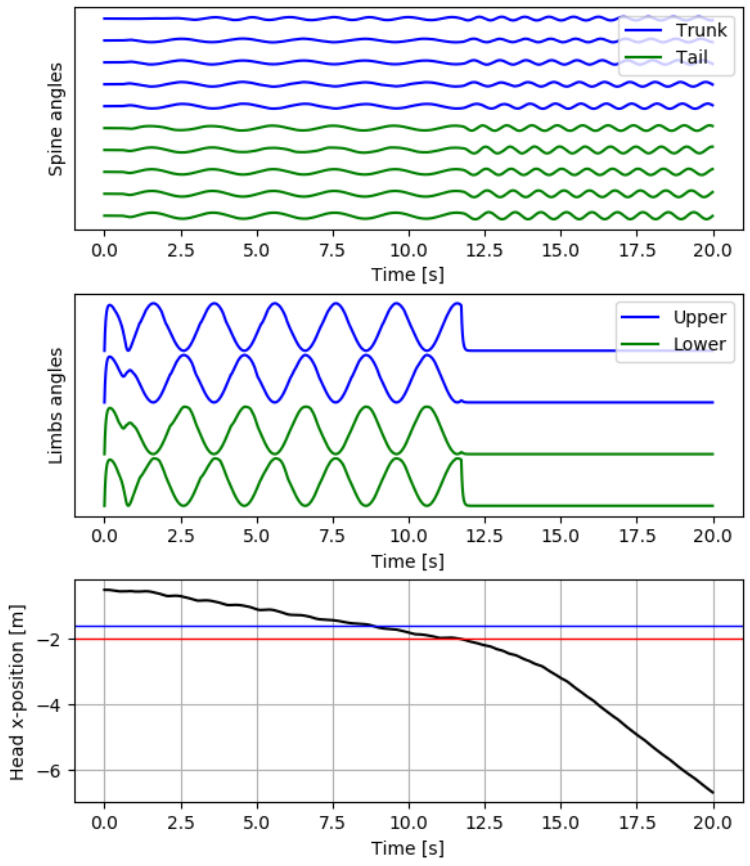


Figure 20: Transition from land to water: activity of the CPG model. (1) Spine angles. (2) Limbs angles. (3) Head x-position. The blue horizontal line corresponds to the water edge (x = -1.6m) and the red one to the head position determining the increase of the drive from 2.5 to 4.5 (x = -2.0m). Simulation parameters used: duration = 20s, amplitude gradient = [0.2, 0.4].

The x-coordinate of the frontier between the land and the water is approximatively x = -1.6m. However, in order to have a gait transition a smooth as possible, it was decided to increase the drive from 2.5 to 4.5, so to switch from a walking gait to a swimming gait, when most of the salamander body was already in the water (corresponding to a head x-position of -2.0m).
As shown in figure 20, when the salamander head reaches x = -2.0m, the limbs stop moving and the spine angles transition from standing waves to traveling waves, as explained in question 8a.

3. (BONUS) Achieve water-to-land transition. Report spine and limb angles, the x-coordinate of the GPS and record a video.

   Hint: Use the record options as shown in exercise_example.py to easily record videos.

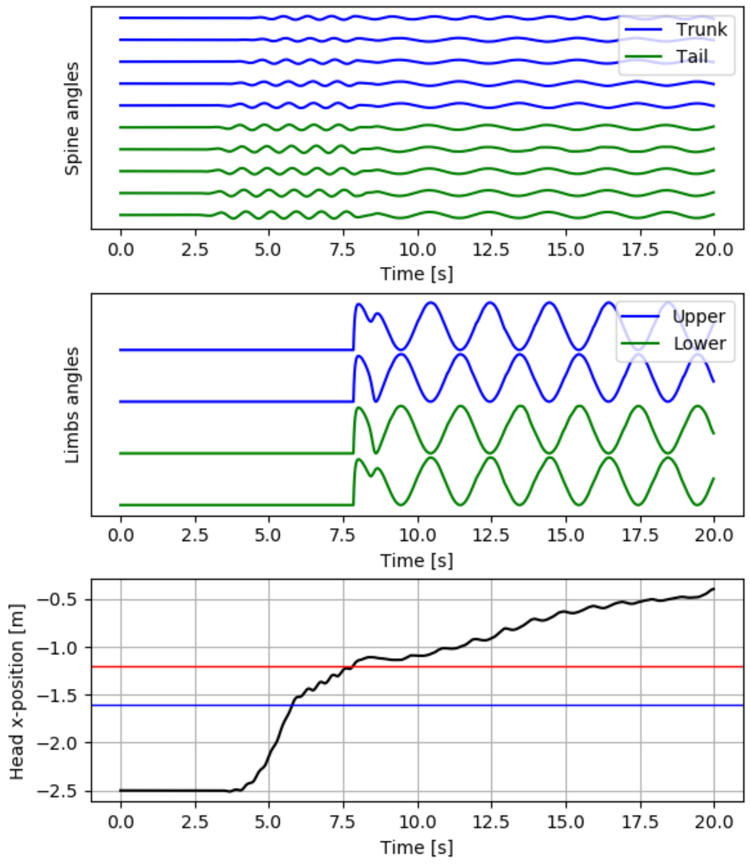


Figure 21: Transition from land to water: activity of the CPG model. (1) Spine angles. (2) Limbs angles. (3) Head x-position. The blue horizontal line corresponds to the water edge (x = -1.6m) and the red one to the head position determining the decrease of the drive from 4.5 to 2.5 (x = -1.2m). Simulation parameters used: duration = 20s, amplitude gradient = [0.2, 0.4].

As before, the goal was to have a gait transition a smooth as possible. So it was decided to switch from a swimming gait to a walking gait, by decreasing the drive from 4.5 to 2.5, when most of the salamander body was already out of the water (corresponding to a head x-position of -1.2m).

Figure 21 shows that when the salamander head reaches x = -1.2m, the limbs start moving and the spine angles transition from traveling waves to standing waves.

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.