Student names: . . . (please update)

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 18-05-2022 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 MuJoCo physics engine. You have an opportunity to use what you've learned until now to make the robot swim and walk in open and closed loop scenarios. 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], [3] and [4]. It is strongly recommended to review [3], [4] and their supplementary material provided on the Moodle. You will be tasked with replicating and studying the Central Pattern Generator (CPG) network proposed in those papers.

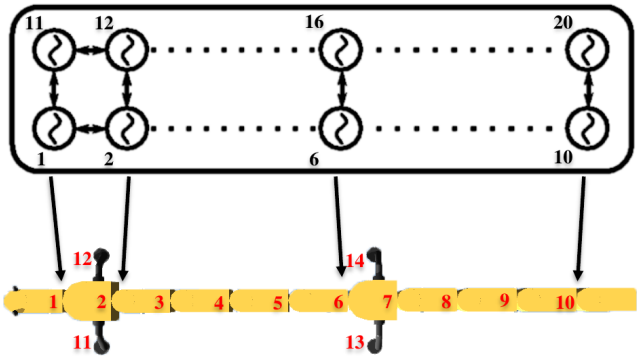

Figure 1: A double chain of oscillators controlling the robot's spine.

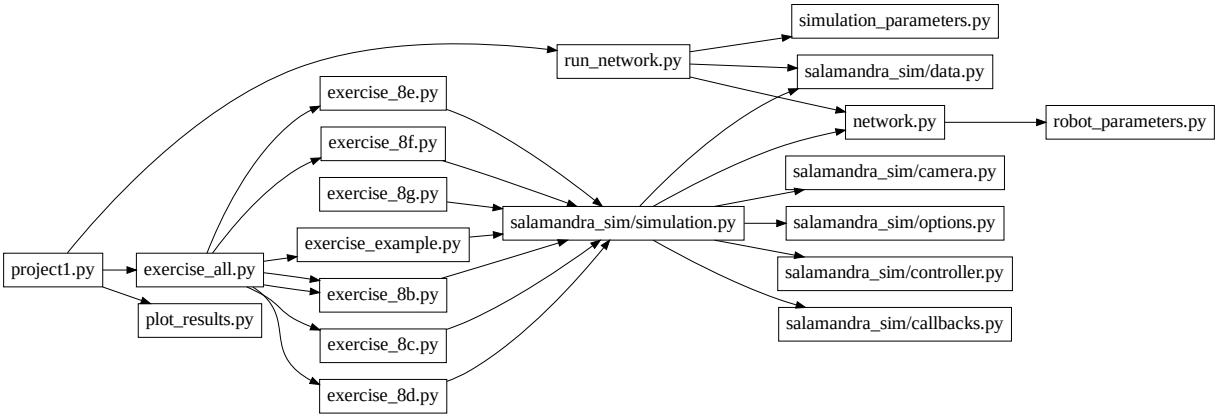

Figure 2: Exercise files dependencies. In this lab, you will be modifying `run_network.py`, `network.py`, `robot_parameters.py` *and* `simulation_parameters.py`

Code organization

- **project1.py** - A convenient file for running the entire project. Note you can also run the different exercises in parallel by activating `parallel=True`. *You do not need to modify this file.*
- **exercise_all.py** - Another convenient file for running all or specified exercises depending on arguments provided. *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 robot_parameters.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 the SimulationParameters class in simulation_parameters.py and provided in exercise_#.py to help you control the values (refer to example.py).
- **simulation_parameters.py** - This file contains the SimulationParameters class and is provided for convenience to send parameters to the setup of the network in network.py::SalamandraNetwork via the robot parameters in robot_parameters.py::RobotParameters. The SimulationParameters is also intended to be used for experiment-specific parameters for the exercises. All the values provided in SimulationParameters are logged for each simulation, so you can also reload these parameters when analyzing the results of an experiment.
- **run_network.py** - By running the script from Python, MuJoCo 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 MuJoCo simulation takes more time.
- **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 **project1.py** for an example on how to call it. *You do not need to modify this file.*
- **plot_results.py** - Use this file to load and plot the results from the simulation. This code runs with the original example provided and provides examples on how to collect the data.
- **salamandra_simulation folder** - Contains all the remaining scripts for setting up and running the simulation experiments. *You do not need to modify any of these file but should still go through them to get a better understanding of the code.*

Prerequisites

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

Pull 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
```

To install the simulator, please follow the instructions in Lab6.

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. Note that it is still possible to proceed with exercise 8a if the simulator is not installed yet.

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 its capability to perform swimming. You will then extend the network to account for the presence of exterosensory feedback due to the hydrodynamic forces acting on the body of the network. You will show that a chain of uncoupled oscillators with sensory feedback is still able to reproduce a travelling wave. You will also study the interplay between open loop and closed loop behavior to optimize the swimming gait according to speed and energetic cost. Finally, you will study the robustness of the different configurations with respect to disruptions of couplings, oscillators and sensors. Exercises 8b to 8g will be performed with a Salamandra robotica II model using the MuJoCo simulation.

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

Salamandra robotica has 8 joints along its spine and 1 joint for each limb. The controller is defined as

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

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

$$q_i = r_i(1 + cos(\theta_i)) - r_{i+8}(1 + cos(\theta_{i+8})) \text{ if body joint} \tag{3}$$

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 ($i = [0, ..., 15]$) and ignoring the leg oscillators ($i = [16, ..., 20]$). Refer to salamandra_simulation/data.py::SalamandraState 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 40 where the first 20 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.

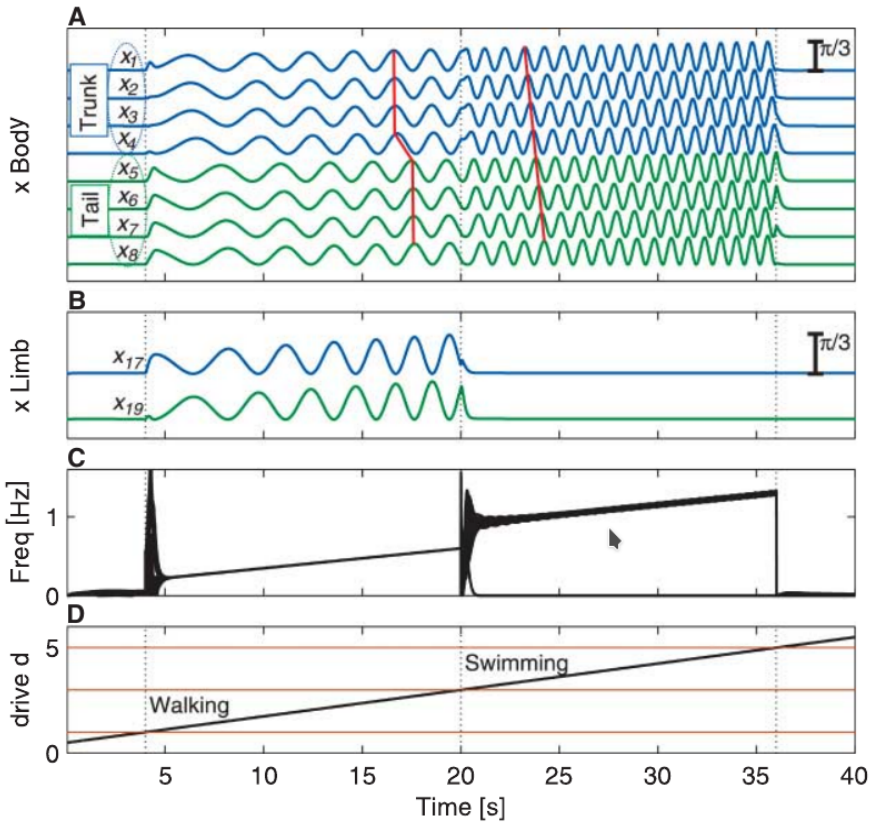

Figure 3: Oscillator patterns from [3], see [3] for details

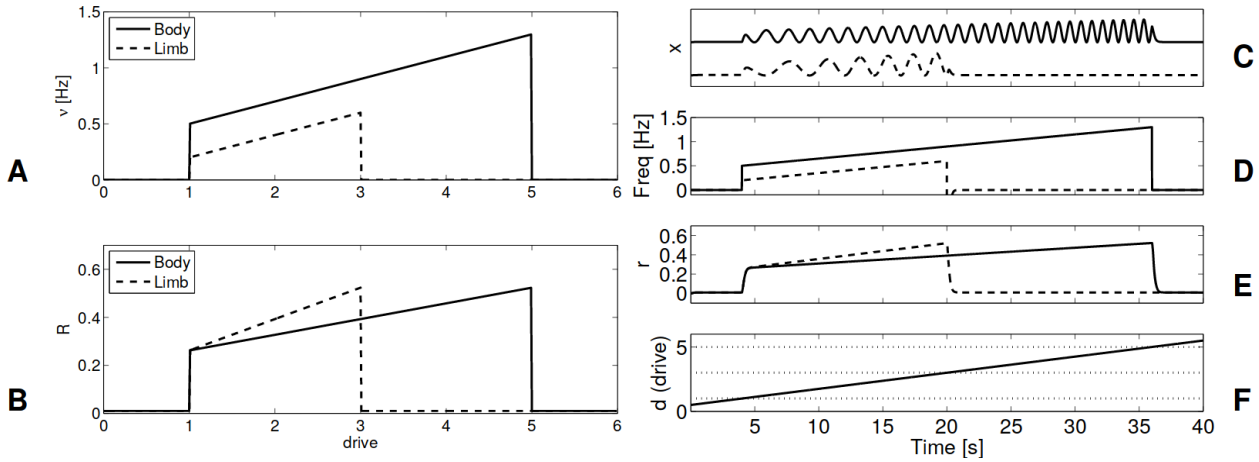

Figure 4: Oscillator properties from [3] supplementary material, see [3] for details

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 sim-

ulation_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 of 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 a justification for the one chosen.

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?

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

8e. Exterosensory feedback

So far we considered constant undulation amplitudes obtained as a result of an open loop control from the central pattern generators. Taking inspiration from [4], you can extend the current model to account for the effect of feedback due to the hydrodynamic forces acting on the body during swimming. The model can be extended as follows:

$$\dot{\theta}_i = 2\pi f + \sum_j r_j w_{ij} \sin(\theta_j - \theta_i - \phi_{ij}) + w_{fb} F_i \cos(\theta_i) \tag{4}$$

$$\dot{r}_i = a(R_i - r_i) \tag{5}$$

$$q_i = r_i(1 + \cos(\theta_i)) - r_{i+8}(1 + \cos(\theta_{i+8})) \text{ if body joint} \tag{6}$$

where F_i is the normal hydrodynamic force acting on the i-th link and w_{fb} is a gain linking the mechanical and neuronal input.

1. Implement a double chain oscillator model whithout couplings between adjactent segments. Is the network still capable of producing a swimming behavior?
2. Starting from the previous step, include the contribution of sensory feedback. What is its effect on the decoupled network? Is it possible to find a parameters' configuration that allows for the creation of a characteristic travelling wave? How does the current model perform compared to the open loop one? Which are the differences in terms of speed and power consumptions?

   Hint 1: The couplings between segments and hemisegments can be accessed from Simulation-Parameters.

   Hint 1: The initial phase of the oscillators can be set with the initial_phases argument to simulation.

   Hint 2: The loads on the body can be accessed in controller.py::SalamandraController::step, through self.animat_data.sensors.hydrodynamics.array. The aforementioned array contains all the hydrodynamic loads acting on each link of the model at every timestep. The six loads are ordered as $[F_x, F_y, F_z, \tau_x, \tau_y, \tau_z]$, where the x axis is aligned with the body link and the z axis is pointing upwards.

8f. Exploiting redundancy

We have previously shown the potential of sensory feedback for substituting local connections between adjacent segments. We will now consider a model including both sensory feedback and ascending/descending connections.

1. Perform a grid search with respect to the intersegmental couplings and the strength of the sensory feedback gain, in order to study the performance of the overall network.
2. Describe the results you observed so far. Which network type displayed the best performance? How do you explain the observed results?

   Hint 1: Select values of amplitude, phase bias and drive that are consistent with the consideration you made in the previous exercises.

8g. Robustness to disruptions

The models analyzed so far have been shown capable, with different performance, to display a swimming behavior. We will now analyze their robustness to different neural disruption.

1. Consider the three networks proposed so far (CPG only, Decoupled segments, Combined) and implement different neural disruptions as proposed in [4]. What is their effect on the swimming performance?
2. Which network showed the highest robustness to disruptions? Explain the results you observed and their implications on salamander locomotion.

   Hint: The properties of the controller are defined and updated in robot_parameters.py. You are free to create internal variables that allow to selectively silence the effects of oscillators, couplings and sensory feedback.

Hint: To ensure a fair comparison between the models, start with configurations showing similar performance.

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.

[4] R. Thandiackal, K. Melo, L. Paez, J. Herault, T. Kano, K. Akiyama, F. Boyer, D. Ryczko, A. Ishiguro, and A. J. Ijspeert, "Emergence of robust self-organized undulatory swimming based on local hydrodynamic force sensing," *Science Robotics*, vol. 6, no. 57, p. eabf6354, 2021.