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. Now you have an opportunity to use what you've learned to make the robot swim and eventually walk. In order to do this, you should implement a CPG based controller, similarly to the architecture shown in Figure 1. Note that the robot model used in this project has 8 joints along the body instead of the 10 shown in Figure 1 thus the controller should be implemented accordingly.

The project is based on the research of [1], [2], [3] and [4]. It is strongly recommended to review [3] and its supplementary material provided on 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 first project. You will start by getting familiar with the code and implementing the CPG network and eventually control the robot. While the MuJoCo simulator will not be necessary yet, we recommend verifying you can install and run it before the next session. Please get in contacts with the TAs if you need any assistance. We also recommend that you implement the controller by Thursday 28-04-2022 such that you can proceed to the simulation experiment during the next session.

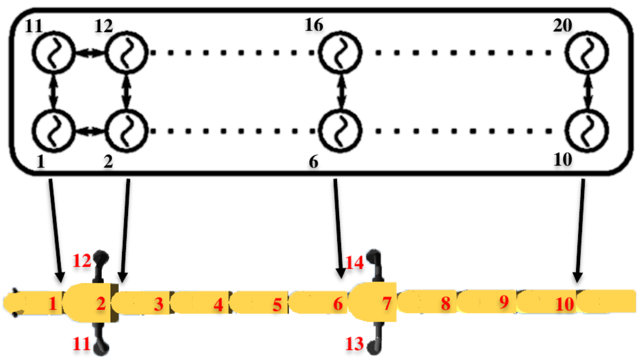

Figure 1: A double chain of oscillators controlling the robot's spine.

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.*

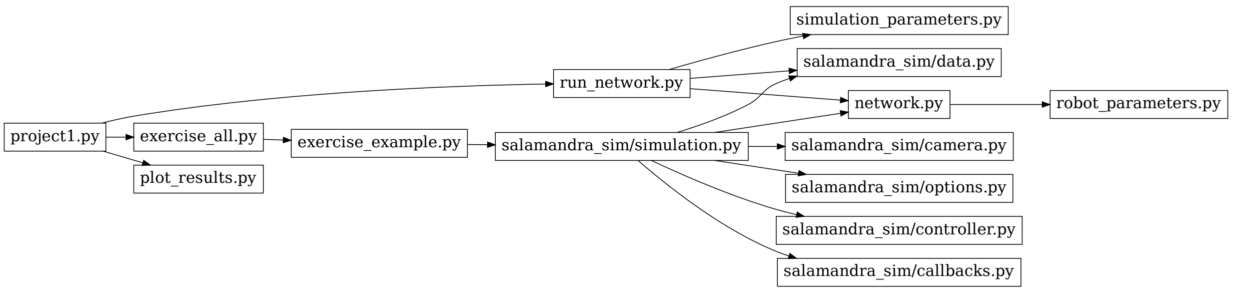

Figure 2: Exercise files dependencies. In this lab, you will be modifying `run_network.py`*,* `network.py`*,* `robot_parameters.py` *and* `simulation_parameters.py`

- **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 **lab8.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.*

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 the swimming, walking and transition behaviors in the Salamandra robotica II model using the MuJoCo simulation and complete the exercises that will be distributed next time.

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 20 elements correspond to the amplitude r_i. The initial state is set in the init of network.py::SalamanderNetwork and is random by default.

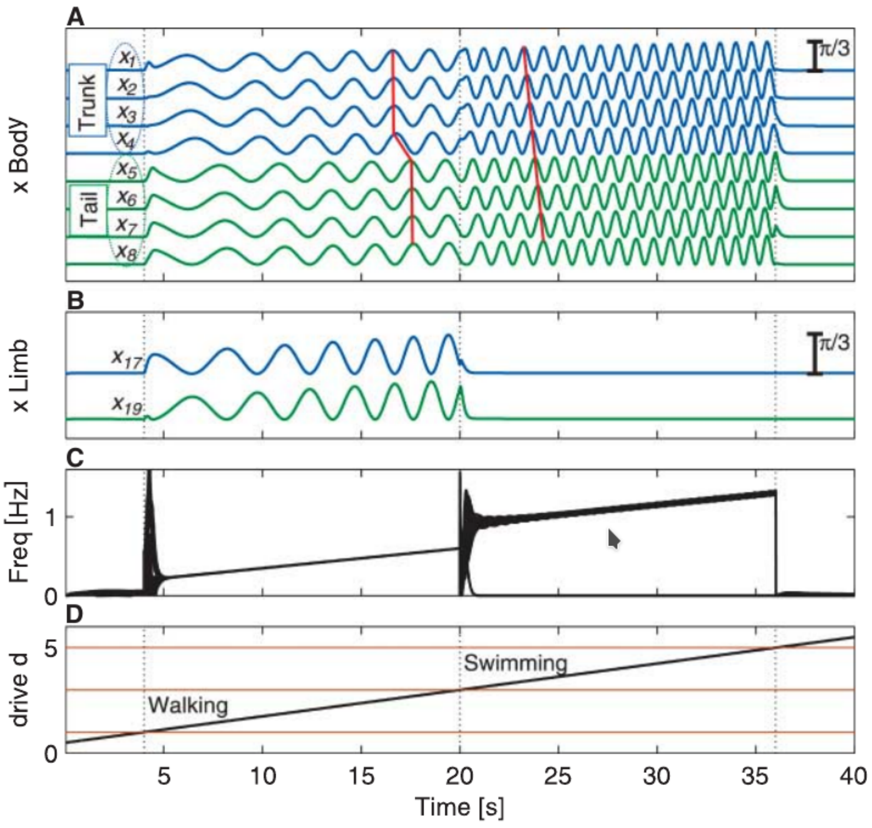

Figure 3: Oscillator patterns from [3], see [3] for details

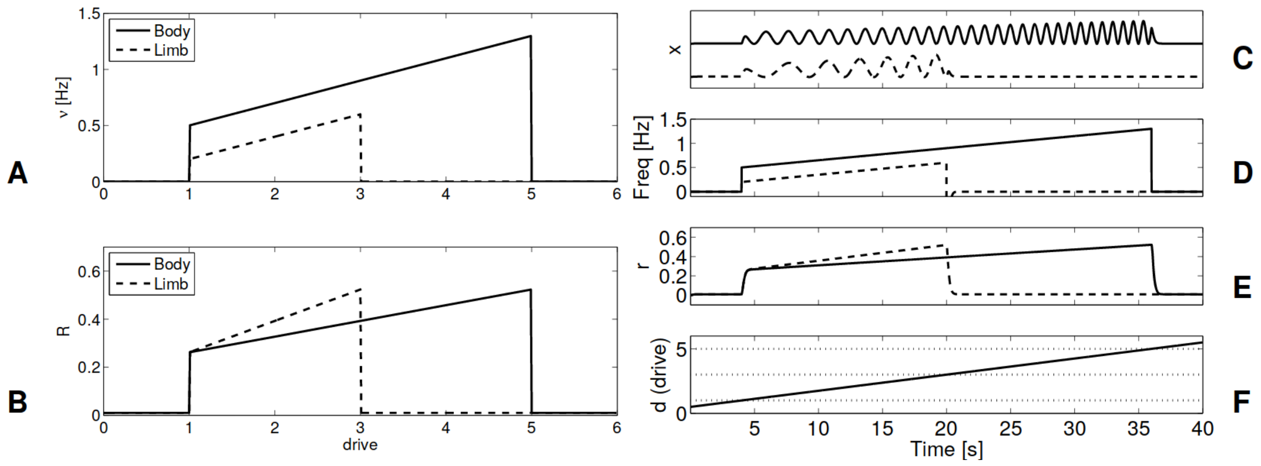

Figure 4: Oscillator properties from [3] supplementary material, see [3] for details

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.