Student names: . . . (please update)

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.

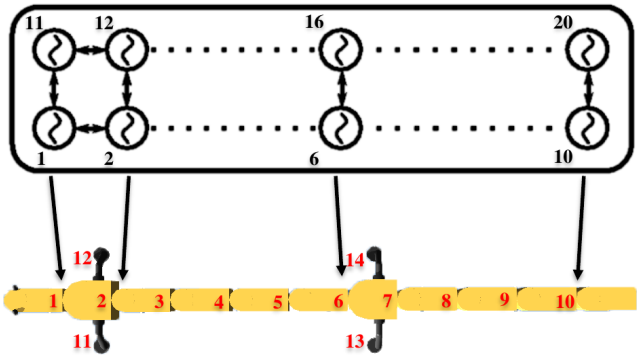

Figure 1: A double chain of oscillators controlling the robot's spine.

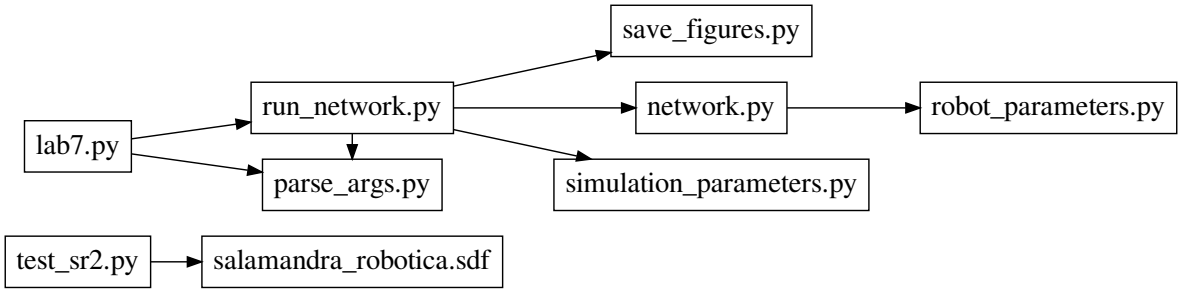

Figure 2: Exercise files dependencies. In this lab, you will be modifying `run_network.py`, `network.py`, `robot_parameters.py` and `simulation_parameters.py`

Code organization

- **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.
- **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).
- **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 7a 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 starting the Webots simulation with physics 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 files.*
- **save_figures.py** - Contains the functions to automatically detect and save figures. *You do not need to modify this files.*

Prerequisites

Make sure you have successfully installed Pybullet by following the instructions outlined in **pybullet_introduction.pdf**. The script **test_sr2.py** will allow you to verify by loading and visualising the robot. Please refer to the TAs if you face any issue with running this file.

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 7a describes the questions needed to implement the oscillator models. After completing exercise 7a you should have an oscillator network including both the spinal CPG and limb CPG.

The remainder of the exercises will be distributed next week.

7a. 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. 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.

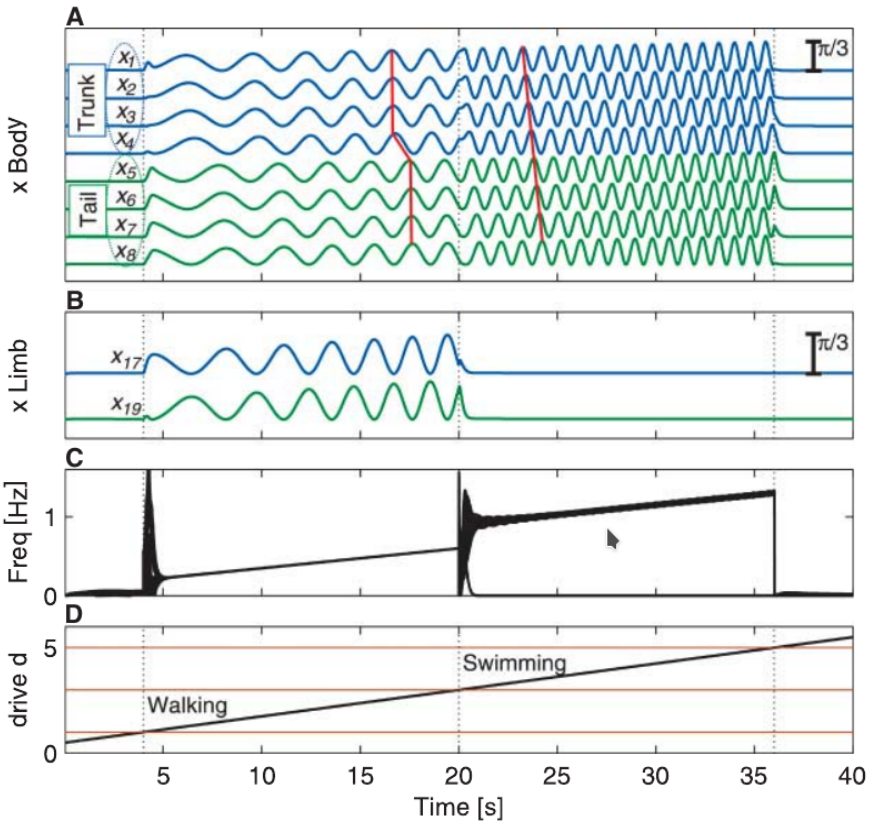

Figure 3: Oscillator patterns from [3], see [3] for details

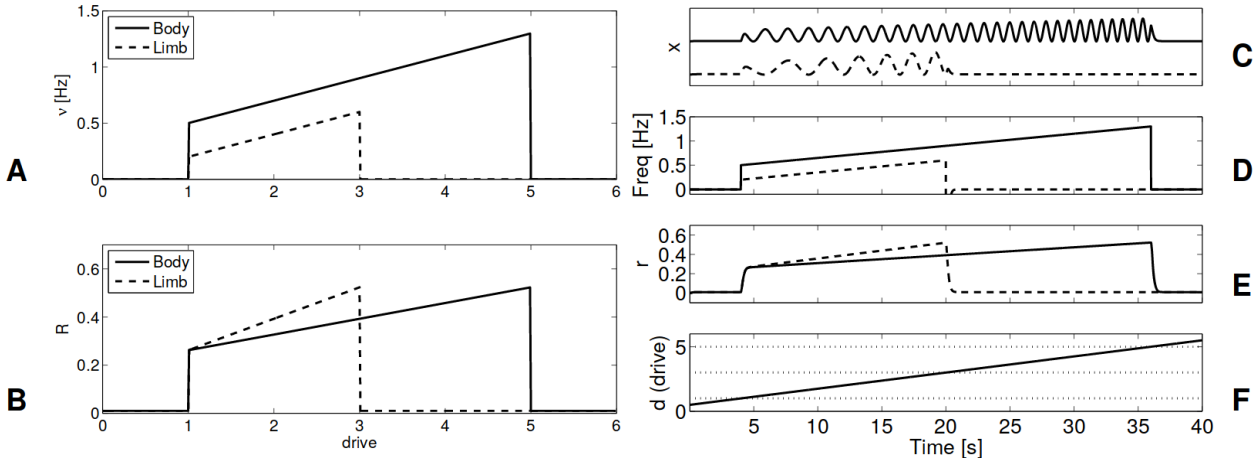

Figure 4: Oscillator properties from [3] supplementary material, see [3] for details

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.