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 08-05-2019 Midnight****.*
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!*

NOTE : The following exercises on Salamandra Robotica are based on the research of [1] and [2].

Swimming with Salamandra Robotica – CPG Model

In this exercise you will control a salamander-like robot Salamandra Robotica for which you will use Python and the dynamics simulator Webots. 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.

In the folder Webots you will find subfolders containing the simulated world file and Python codes describing the controller. Do not change the relative positions of files within those folders.

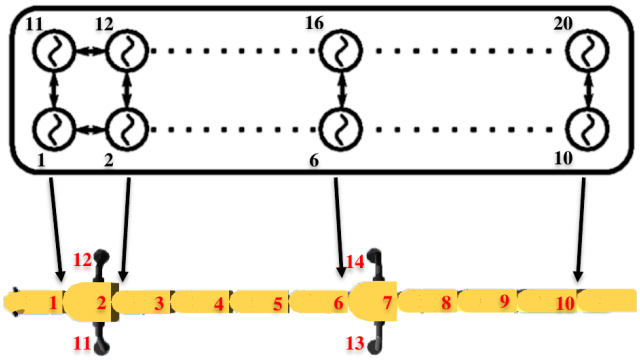

Figure 1: A double chain of oscillators controlling the robot's spine.

Code organization

- **Webots::worlds::cmc_salamander.wbt** - This is the world file which describes the world and allows to run the simulation. You can run a simulation by running this file with Webots. It also automatically loads the pythonController.
- **Webots::controllers::pythonController::pythonController.py** - The main robot controller is implemented in pythonController.py. This file calls classes and functions from other files to control the robot and for logging. There is also a run_simulation function which is provided for convenience to easily run multiple simulations with different parameters. Note that network parameters can be provided here.
- **Webots::controllers::pythonController::cmc_robot.py** - Contains the SalamanderCMC class, which is used for controlling and logging the robot. Feel free to modify this file to extend

the logging capabilities. Note that the logging makes use of Numpy.savez to save the data.

- **Webots::controllers::pythonController::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.
- **Webots::controllers::pythonController::solvers.py** - This features fixed time-step solvers which will can are used by network.py for solving the ODE at each timestep. Feel free to switch between the Euler and the Runge-Kutta methods. *You do not need to modify this files.*
- **Webots::controllers::pythonController::run_network.py** - By running the script from Python, Webots 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 analyse its behaviour. This is useful for debugging purposes and rapid controller development since starting the Webots simulation with Python in the loop takes a bit of time.
- **Webots::controllers::pythonController::plot_results.py** - Use this file to load and plot the results from the simulation. This code runs with the original pythonController provided.
- **Webots::controllers::pythonController::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.*
- **Webots::controllers::pythonController::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 Webots by following the instructions outlined in Lab 7

Complete the tutorial and practice examples of Webots as outlined in Lab 7

Open the `Webots::worlds::cmc_salamander.wbt` file in Webots. This should launch the Salamandra Robotica model in simulation world

Running the simulation

Now when you run the simulation, the Salamandra Robotica model should float on the water with no errors in the Webots console dialog. At this point you can now start to work on implementing your exercises.

Questions

8a. Implement a double chain of oscillators

Salamandra Robotica has 10 joints along its spine. Implement a double chain (dynamics described in equations 1 and 2) of oscillators in a way that lateral oscillator pairs are assigned to a single joint (see Figure 1).

NOTE : No need to implement leg oscillators for this exercise.

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

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

$$\dot{q}_i = r_i(1 + cos(\theta_i)) - r_{i+10}(1 + cos(\theta_{i+10})) \tag{3}$$

with θ_i the oscillator phase, f the frequency, w_ij the coupling weights, ϕ_{ij} the nominal phase lag, r_i the oscillator amplitude, R_i the nominal amplitude, a the convergence factor and q_i the spinal joint angles.

1. Implement the double chain oscillator model using the functions `network.py::phases_ode` and `network.py::amplitudes_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 set your coupling weight parameters in the function `network.py::AmplitudeEquations::set_parameters` and set your convergence rates and desired amplitudes in `network.py::PhaseEquations::set_parameters`
2. Use 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 first in Python(`run_network.py`) and then in Webots that your code works as expected. Use the functions in `plot_results.py` to report your spinal joint angles q_i.

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

How does phase lag and oscillation amplitude influence the speed and energy? Use the provided pythonController.py::run_simulation() 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. Feel free to extend the logging in cmc_robot.py to show additional measurements if necessary. Include 2D/3D plots showing your grid search results and discuss them. How do your findings compare to the wavelengths observed in the salamander? Run the grid search twice, for frequencies of 1Hz and 2Hz.

- **Hint 1:** To use the grid search, check out pythonController.py::run_simulation(). This function takes the desired parameters as a list and runs the simulation. An example of parameter sweep is shown in pythonController.py::main() as an example. The function will repeatedly run simulations, each time with a different combination of parameters from the parameters list. The results are logged as simulation_#.npz in a specified log folder. After the grid search finishes, the simulation will close, you can remove this feature by commenting world.simulationQuit(0) in pythonController.py::main().

- **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 Webots in a fast mode.
- **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.

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, modify the function network.py::AmplitudeEquation::set_parameters() such that the input variable amplitudes=[Rhead, Rtail] now has the size 2, and within the function you interpolate the amplitude gradient between values Rhead and Rtail. Note that you can the provide this amplitudes parameter from pythonController.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 Webots 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.

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.