Developing a Walking Controller for a Three-link 2D Biped

Jessica Lanini, Andrea Di Russo, Dimitar Stanev, Laura Paez,
Vivek Ramachandran, Mehmet Mutlu and Auke Ijspeert
{jessica.lanini - andrea.dirusso - dimitar.stanev - laura.paez - vivek.ramachandran - mehmet.mutlu}@epfl.ch

Start: 24/09/2019 - Deadline: 20/12/2019

Introduction

I am sure by now you have seen many legged robots. The question is, if I give you a legged robot how would you make it walk? Where do you start? Of course, there is an interface where you can receive sensory data and send commands, but will you start by sitting and sending commands and see what happens? No. So, what is the starting point? The starting point is to model, then design control and finally simulate. The whole point of the mini-project is to realize the following **Control Design Pipeline**:

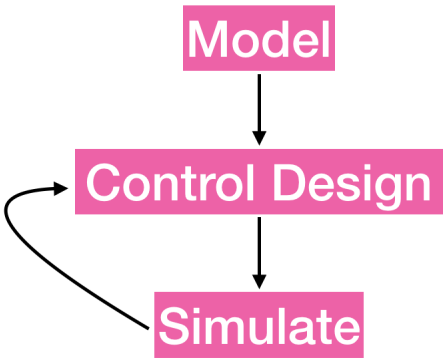

Figure 1: **Control Design Pipeline**

In order to understand the main principles behind the control design, we will work with a simple model: the **Three-link 2D Biped**, as represented in Fig. 2.

The project will be divided into three main parts:

- Modelling and visualization of the 3-link;
- Solving the equations of motion of the 3-link biped (simulation);
- Design of two different walking controllers, evaluate the resultant gaits and compare the performances.

You are asked to **develop these three parts and report your methods, results and observations in a final report**.

You are asked to structure your report according to the following sections which will be evaluated with the corresponding percentages:

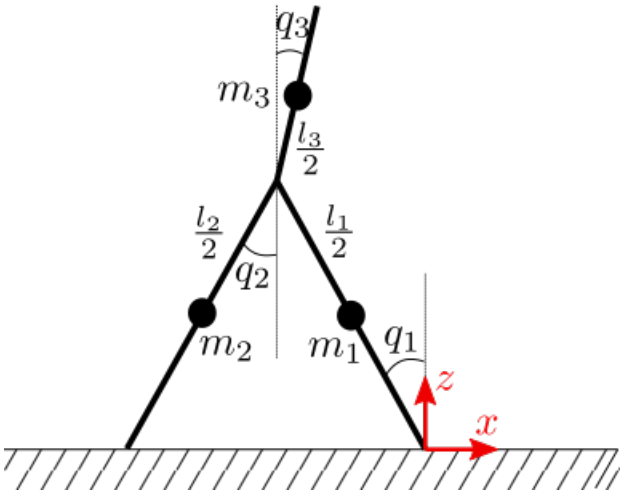

Figure 2: **Three Link Biped**

- Introduction : 3%
- Methods : 5%
- Results : 40% (kinematics and dynamics: 10%; controllers: 30%)
- Discussion : 50%
- Conclusion : 2%

In this lecture and in the next you will complete the second part: *Solving the equations of motion of the 3-link biped (simulation).*

Exercise 3.1: Equations of motion

In this part you are asked to complete the following matlab files (in the "solve_eqns" folder):

- eqns.m
- event_func.m
- solve_eqns.m

Complete eqns.m

We would like to solve the equation of motion:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = Bu$$

with ode45 solver of MATLAB. The signature of the ode45 solver is as follows:

$$[T, Y, TE, YE] = ode45(@eqns, tspan, y0, options)$$

where eqns.m is a MATLAB function with the signature dy = eqns(t, y) representing the state-space form of the equations of motion above.

First off, run the script set_path.m to add the required directories to your path. Then start off by completing eqns.m in the solve_eqns folder. Remember, you need to write the equations of motion $M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = Bu$ in the form $\dot{y} = eqns(t, y)$. You can do this in many different ways but for consistency use this definition of y as follwos:

$$y = [q, \dot{q}]$$

Complete solve_eqns.m

As mentioned above, the signature of the ode45 solver is:

$$[T, Y, TE, YE] = ode45(@eqns, tspan, y0, options)$$

where eqns.m is the equations of motion in the state-space form, tspan is the time span for which you would like to solve the ode, y0 is the initial condition and options defines the options for the ode solver. The impact map which is an event function is defined using the options.
In particular with **odeset** set the options in solve_eqns.m so that the relative tolerance is 1e-5 and the event function is event_func . Then complete the event function event_func.m . We want the event function to trigger when the swing foot hits the ground. To this end, set the value in event_func.m appropriately. Additionally, we only want to declare an event if the swing foot hits the ground with a negative z velocity. To account for this, set the direction in event_func.m appropriately. Run the simulation and animate the results for different initial conditions to verify your animation intuitively.

Exercise 3.2: Animate

In this second part you are asked to complete the following matlab file:

- animate.m (in the "visualize" folder)

If Y is the solution to the ode, at each time step i you can extract the angles q by q = Y(i, 1:3) . Use this q as the input of the visualize.m function to animate the solution Y of the equations of motion. Note that the visualize.m function has an extra input r0 which is the position of the stance foot in the global frame. Calculate the real time factor as defined in the animate.m script.
Finally answer to the following questions:

- In the animations, what does a real-time factor of 1 mean? How about a real-time factor less than 1?
- How does 'skip' in animate.m effect the real-time factor and the speed of the animation?
- What is the role of 'r0' in animate.m?

NB: please include the answers to these questions in your report!

Exercise 3.3: Test your results!

Make sure that your results for the previous parts are correct!
To this end please compare your *sln* structure given as output by **solve_eqns.m** with the provided *sln_test* with the *isequaln* function in matlab.