Simulation of a three-link 2D biped

Dimitar Stanev, Andrea Di Russo, Laura Paez, Alireza Manzoori Mohammad Askari, Arthur Bricq, Auke Ijspeert

{dimitar.stanev/andrea.dirusso/laura.paez/ali.manzoori/mohammad.askari/arthur.bricq/auke.ijspeert}@epfl.ch

Introduction

In this exercise, we will use the model developed so far to perform simulations. This means that we will explain how to transform the equations of motion, which are of second order, into first order differential equations that can be solved numerically. Numerical integration methods are commonly used for solving differential equations that might not have a close form solution, provided initial conditions. This is the case for the swing phase model that we developed. What about the discrete events due to the impact with the ground? This could be handled by implementing an event function that could check and interrupt the integrator when a condition is met. Once the numerical integrator is interrupted, we can update the initial conditions and proceed with the numerical integration to the next gait cycle.

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:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{B}\boldsymbol{u}$$

with *ode45* solver of MATLAB[1]. 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 $\frac{d\boldsymbol{y}}{dt} = eqns(t, \boldsymbol{y})$ representing the state-space form of the equations of motion above.

First off, please run the script *set_path.m* to add the required directories to your path. Then, start by completing *eqns.m* in the "solve_eqns" folder. Remember, you need to write the equations of motion in the first order differential equations form ($\dot{\boldsymbol{y}} = eqns(t, \boldsymbol{y})$). You can do this in many different ways, but for consistency use this definition of $\boldsymbol{y}$ as follows:

$$\boldsymbol{y} = [\boldsymbol{y}_1, \boldsymbol{y}_2]^T = [\boldsymbol{q}, \dot{\boldsymbol{q}}]^T$$

then, it follows that:

[1] https://www.mathworks.com/help/matlab/ref/ode45.html

$$\dot{\boldsymbol{y}} = \Big[\boldsymbol{y}_2, \boldsymbol{M}^{-1}(\boldsymbol{y}_1)\Big(\boldsymbol{B}\boldsymbol{u} - \boldsymbol{C}(\boldsymbol{y}_1, \boldsymbol{y}_2)\boldsymbol{y}_2 - \boldsymbol{G}(\boldsymbol{y}_1)\Big)\Big]^T$$

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* (*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[2]. Then, complete the event function *event_func.m*. We want the event be triggered 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. **Please read the MATLAB documentation for *ode45* and event functions that are provided in the footnotes.**

Exercise 3.2: animate

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

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

If $\boldsymbol{Y}$ is the solution to the ode, at each time step i you can extract the angles $\boldsymbol{q}$ by $\boldsymbol{q} = \boldsymbol{Y}(i, 1:3)$. Use this $\boldsymbol{q}$ as the input of the *visualize.m* function to animate the solution $\boldsymbol{Y}$ of the equations of motion. Note that the *visualize.m* function has an extra input $\boldsymbol{r0}$ 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 $\boldsymbol{r0}$ in *animate.m*?

Please include the answers to these questions in your final 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.

Exercise 3.4: further reading

Please examine the *numerical_integration.zip*, which contains theory and examples on the topics of numerical integration. Its intention is to give you more details on the different numerical integration methods as well as highlight some of the problems that are commonly encountered (e.g., numerical integration stability, accuracy and adaptive time stepping). The examples are written in Python Jupyter notebook format. You can either evaluate the notebook (*.ipynb*) using Jupyter or directly open the *.html* file in your browser and read its content. **Please include the answers to the questions raised in this example in your final report!**

[2] https://www.mathworks.com/help/matlab/math/ode-event-location.html