Semester Project Report

Real-time control of a humanoid robot: Dynamic walking using a muscle-reflex model

Supervisor:
Jesse van den Kieboom

Authors:
Allan Barrea
Nicolas Van der Noot

Professors:
Renaud Ronsse [UCL]
Auke Jan Ijspeert [EPFL]

January 10, 2013

Contents

Chapter 1

Introduction

The need for humanoid robots

In our society, there is an increasing need for humanoid robots. This kind of robots try to reproduce several human features and integrate them in a single machine. There has been several famous examples in the past decades, as covered in detail in [SK08]. Two examples we will come back to later are ASIMO, the multi-purpose research robot made by Honda, and the infant-like iCub project, developed at the Italian Institute of Technlogy (IIT) and funded by the European Commission.

Humanoid robots are a way to better understand humans. Indeed, as explained in [Ron12], one can use the robots to understand the principles behind the human capabilities. There are lots of areas in which that idea applies: communication, learning, physical exploration of the world, etc. In this report, we look at the locomotion aspect of these humanoid robots.

Humanoid robot locomotion

When we consider human locomotion, which seems very complex, we can ask ourselves the following question: "Why didn't the evolution give us wheels?". Indeed, a mobile robot equipped with wheels seems much easier to control than one with legs. Therefore, legged locomotion must have some key advantages compared to wheeled locomotion.

One of these advantages is that legged locomotion allows for a better handling of rough terrains. Indeed, a wheeled robot is in trouble when faced with stairs, rubble or that kind of uneven terrain. The reason is that wheels (or caterpillars) always need a continuous support from the ground. Legs, however, allow a robot to walk even with a discrete ground support. Another advantage is that a legged robot can potentially be lighter than a wheeled one, since there is no need to have a frame to support wheels.

At this point, we have to separate between two kinds of humanoid walking. On the one hand, there is static walking, where the robot remains stable at every moment of the motion. On the other hand, there is the dynamic walking, where the robot falls if it is frozen in the middle of the motion. With this kind of gait, the stability can only be achieved using an active control of the leg motion. Therefore, dynamic walking requires to use a dynamical model of the robot whereas static walking is achievable considering only the kinematics of the robot.

In everyday life, humans walk dynamically and not statically. Dynamic walking looks much more natural than static walking. Furthermore, if the evolution favoured dynamic walking, it must be more energy-efficient than static walking. In this report, we focus ourselves on the implementation of a gait controller to achieve dynamic walking with a humanoid robot.

Using a muscle-reflex model to generate human-like walking gaits

Now, getting back to the introductory question for the last paragraph, we wonder whether the control of a legged walking gait is really so difficult to achieve or if there is some way to reduce this apparent complexity. It seems that the properties of the legs play a major role in the stability of the gait, as summarized in [GH10].

In the same article, Hartmut Geyer provides a controller to make a humanoid walk dynamically. It is based on a muscle-reflex model, i.e. it uses a set of contracting muscles activated by several reflex rules to produce the torques applied on the sagittal joints of the robot legs. So, it is a two-dimensional human-like gait controller.

This controller has already been implemented in simulation by previous students at Biorob, namely Steve Berger [Ber11] and Florin Dzeladini [Dze13]. Our project is about merging the reality gap between previous simulations and a real state-of-the-art robotic platform.

From simulation to reality

The main difficulty we had to cope with was to implement this model on a real robot, which brings several problems. First, the robot has a real body, with e.g. moving arms whereas the upper body of the robot in the original Geyer's model was simply modelled by a single Head-Arms-Trunk (HAT) rigid body. Furthermore, the actual robot segments do not have the same lengths and masses as in Geyer's model. A scaling of the model parameters is then required. Particularly, the relative size of the real robot foot with respect to its body's height is bigger than the one in Geyer's model. Therefore, the gait obtained looks different (and, for the moment, less natural) than the one in Geyer's model.

Another major difference between the simulated version of the robot in Geyer's model and the real one is that the real robot has actuators to apply torques whereas the ideal version assumed that the computed torque references were directly applied on the robot joints. Consequently, there are additional errors introduced with the real robot due to the non ideal torque controllers.

International context of the project

This project is part of an ongoing international collaboration. Indeed, the robot we use for this project, the CoMan, has been designed and built by the Italian Institute of Technology (IIT, Italy). It is based on the iCub, developed at IIT, too. The simulation environment used to implement the dynamic simulator of the robot has been developed at the Université catholique de Louvain (UCL, Belgium). This is the Robotran simulation environment. Finally, the Biorob Laboratory (EPFL, Switzerland) has bought the latest release of the robot and possesses a strong expertise in developing locomotion controllers for humanoid robots. Our project is shared between UCL and EPFL. This report overviews the first part of the project, carried out at EPFL during our Erasmus stay.

Results achieved

To realize this project, we first designed the gait controller on the robot model. Then, we performed some preliminary tests with this controller on the real robot. However, we lacked time to do a full validation of the controller on the real robot. This lack of time can be explained on the one hand because the robot arrived at Biorob lately in the semester and, on the other hand, we had to develop the robot model in addition to the gait controller, which took us a lot of time.

To speed up the transfer of the gait controller, developed in simulation, to the robot (and potentially vice-versa), we organized the simulator code in several layers with well-defined interfaces between them. This way, it is possible to reuse most of the controller files and transfer them without modification to the actual robot.

Report organization

This report is organized in the following way. First, we begin in chapter 2 with the presentation of the tools used to develop the model. Then, we detail the design of the gait controller in chapter 3. Next, we review the development of the robot model (dynamic simulator) in chapter 4. After that, an optimization strategy is described in order to find the controller parameters allowing the robot to walk dynamically in a robust way (chapter 5). Then, an interface step is needed in order to transfer the work done in simulation to reality, this is covered in chapter 6. Finally, we run some preliminary tests on the real robot and get and analyse the results in chapter 7. At last, we sum up all the work done and the results obtained in the concluding chapter 8.

Chapter 2

Robotic Platform & Tools

As explained in the introductory chapter, the aim of our project is to develop a controller to make a humanoid robot walk dynamically. The robotic platform used in this project is the CoMan, a humanoid robot developed by the Italian Institute of Technology (IIT). The name of the robot stands for COmpliant HuMANoid Platform.

In addition to the controller development, we also had to develop the dynamic simulator of the robot itself to test the controller on this simulator before validating it on the real robot. The detailed development of this simulator is described in chapter 4. In the present chapter, we focus on the description of the tool used to achieve this task, namely the Robotran simulation environment.

2.1 CoMan

2.1.1 Hardware overview

A robot based on the iCub

The CoMan robot is a 23-degrees-of-freedom (DOF) compliant humanoid robot. The figure 2.1 shows a view of the current version of the CoMan, the one used for this project. It is based on the iCub robot[1] (illustrated in figure 2.2) but there are major differences between the iCub and the CoMan. The main one is that the main joints of the CoMan are actuated with series elastic actuators (SEA). This passive compliance provides an increased capacity to withstand shocks. Moreover, this can be useful to achieve safer interaction and potentially energy efficient locomotion by storing potential energy in the SEA elastic elements.

The current version of the CoMan has SEA for the the pitch joints in the legs, the waist, the shoulders and the shoulder roll joints as described in [DMMC$^+$12].

As other differences between the iCub and the CoMan, we can note on the one hand that the CoMan is equipped with a battery, that it is fully covered (there are no exposed wires) and that the torso joint has been redesigned. On the other hand, the current version of the CoMan has neither hands nor head yet. The hands are still under development whereas the head will probably never exist as it is not necessary except to get a body mass distribution closer to the one of an actual child.

Control hardware

The brain of the robot is located in its torso and takes the form of an embedded computer (the PC104) executing a command-line version of Linux.

Each motor is locally controlled by its own low-level controller. A typical motor controller is a small board with a C-programmable DSP and a 100Mbit Ethernet communication interface. This interface allows the system to collect data up to 1 kHz. In the robot, there are also several Ethernet switches to connect the low-level controller to the PC104, as shown in the figure 2.3.

[1]More info on `www.icub.org`.

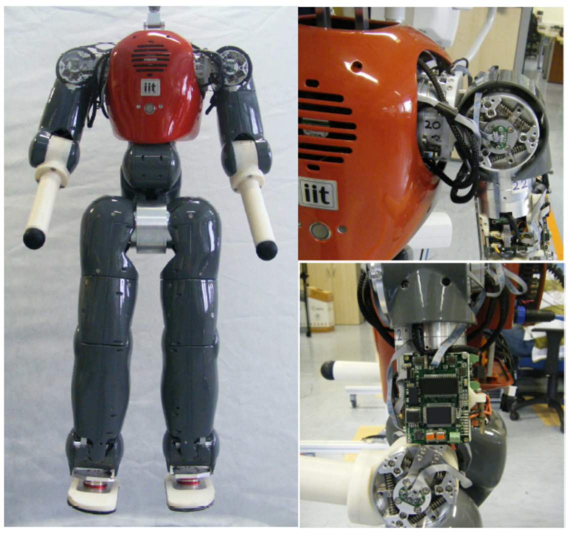

Figure 2.1: CoMan humanoid robot (from [DMMC+12]): [left] the whole robot; [top-right]: SEA of the left shoulder; [bottom-right]: left arm (control board and SEA).

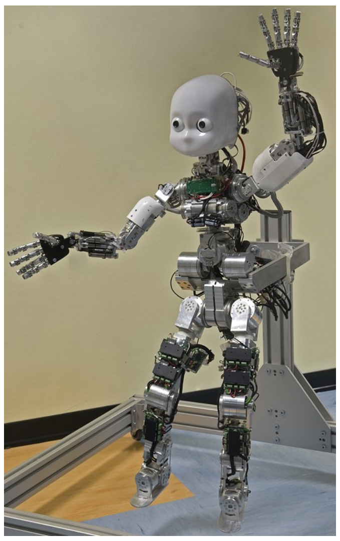

Figure 2.2: iCub robot (from `www.icub.org`).

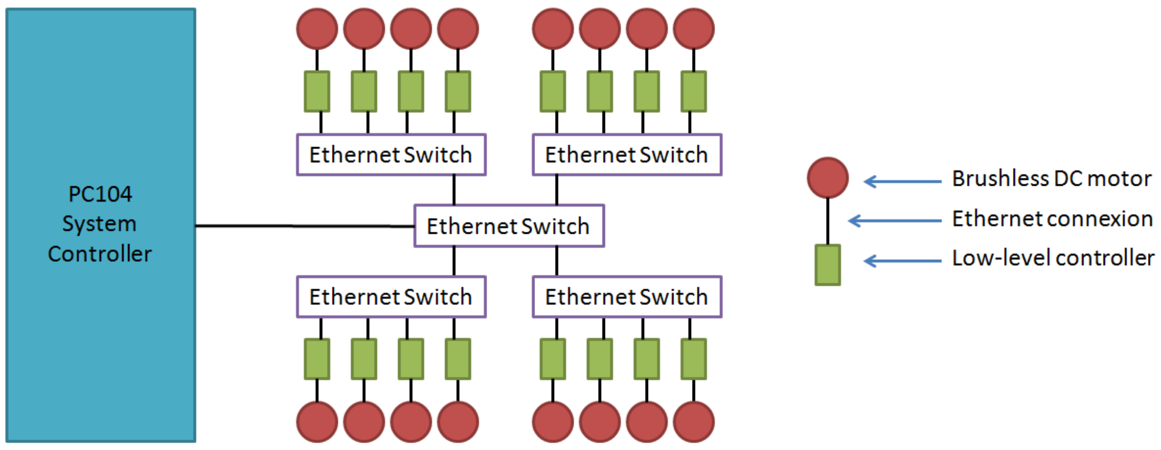

Figure 2.3: Motor controller boards interconnection.

It is possible to control the SEA both in position or in impedance. Indeed, the actuators embed position encoders and torque sensors. On top of that, the CoMan is also provided with 6-axis force/torque sensors on each foot. This allows for sensing precisely the ground reaction forces under the robot's feet, which is a key input for the controller we developed, as will be described in chapter 3.

Further information and references

Further details about the CoMan robotic platform can be found in [DMMC+12], [TMDMC12] and [IIT12]. Besides, [TLSC11] gives more information about the specific development of the lower body of the CoMan by describing the design of the cCub, the intermediate version of the robotic platform between the iCub and the current version of the CoMan.

2.1.2 Model overview

We received from IIT a first version of the dynamic simulator of the CoMan. This version was implemented on Matlab using the Robotran framework and, as will be described in chapter 4, we made lots of modifications on it. Nevertheless, some key characteristics of the model are still present in the version we use now and one of these is the floating base.

As shown in figure 2.4, the dynamic simulator is based on the floating base representation of legged robots. A floating base has its base body free to move rather than being fixed in space. As explained in [DMMC$^+$12], "the free motion of the base is represented by a 6-DOF joint attached between the humanoid robot and the world inertial frame to describe the free motion of the humanoid with respect to the inertial frame. The 6-DOF joint consists of 3 translational joints and 3 rotational joints about the XYZ axes. This formalism unifies all the phases of legged locomotion, including single support, double support and flight phase, as well as the falling phase into one single model. Moreover, it simplifies the simulation models by removing the switches between various phases of walking. Since the floating base joints are unactuated, the ground model has a significant role in the robots balance and locomotion." The ground model is presented in chapter 4.

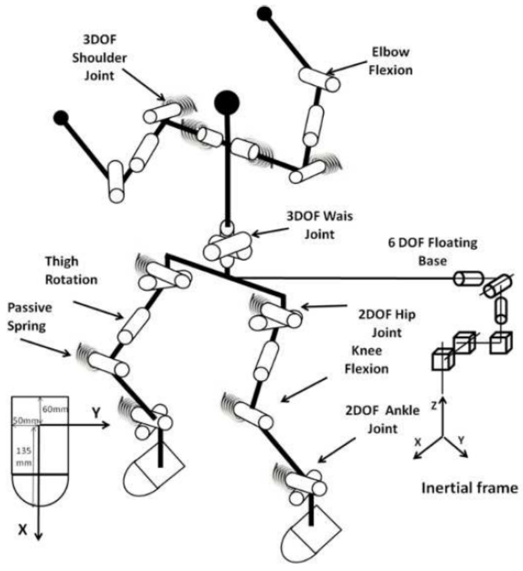

Figure 2.4: CoMan's floating base kinematic model (from [DMMC$^+$12]).

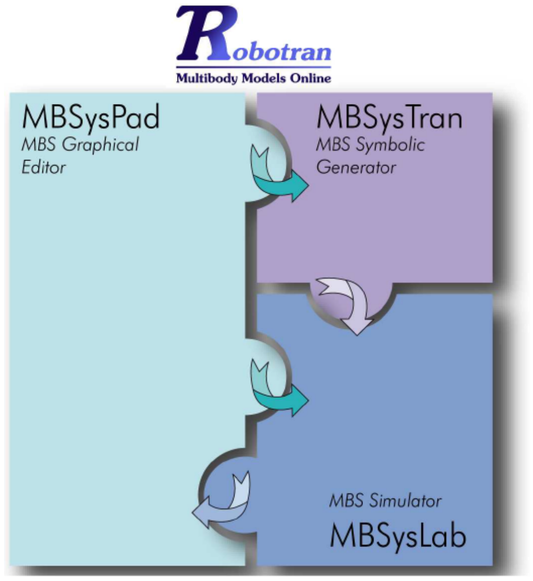

Figure 2.5: Structure of the Robotran simulation environment (from [CER09]).

To introduce the rest of the model, we first need to present the simulation environment on which it is based, namely the Robotran software. This is the topic of the next section.

2.2 Robotran

2.2.1 Overview

What is Robotran?

Robotran is a general purpose multibody simulation environment developed at the Université catholique de Louvain (UCL) within the Centre for Research in Mechatronics (CEREM).

A multibody system is a mechanical system composed of several rigid or flexible bodies interconnected by joints and which can undergo large translational and/or rotational displacements with respect to each other. These bodies are submitted to internal or external forces and torques. Those forces can result from both passive elements such as springs, or active components such as electromechanical actuators.[2]

[2]Adapted from www.robotran.be.

Robotran uses the multibody approach to model the dynamic behavior of such a system. In our case, the CoMan is seen as a set of several rigid bodies interconnected in a tree-like structure (as shown in figure 2.7 which will be described in further details later on). In this structure, each body is attached to only one "parent" body. Besides, each body can have several "child" bodies connected to it as a parent body.

Model development flow

A typical development flow of a multibody system (MBS in short) consists in 4 steps, summarized in figure 2.5.

- **Step 1:** Draw your multibody system, using the **MBsysPad** Graphical Editor.
- **Step 2:** Generate your multibody equations, calling the **MBsysTran** Symbolic Generator.
- **Step 3:** Simulate your multibody model, using the **MBsysLab** Simulator.
- **Step 4:** Analyse your multibody simulation, in **Matlab** and in **MBsysPad Animation**.

These steps will be illustrated in the section 2.2.4 where we present the CoMan model in Robotran.

The key characteristic of Robotran is to generate the equations of motion symbolically, whereas other simulation environments (such as Webots) rely on numerical models, which are more sensitive to numerical errors. In practice, this symbolic equations generation is done on a server at UCL and this is the only step hidden to the user. The rest of the Robotran program is entirely open and the user can modify it at will. Anyway, a free-of-charge access to the UCL server is possible by simple request addressed at the CEREM.

After the symbolic equations generation, these equations are made available to the user who can use them to simulate the dynamic behaviour of the multibody system with MBsysLab. This part of the program consists in several Matlab (Matlab code) or Simulink (C code) files which can be tuned by the user according to the needs.

To assist the user, the Robotran framework provides predefined templates to add actuator dynamics, constraints and external forces on any point of the multibody system, as well as sensors with symbolic expressions to acquire kinematic information such as Jacobian, linear and angular position, velocity and acceleration of any point on the multibody system. Given that, it is e.g. possible to include the constitutive law of an actuator model or to implement a ground contact model, both of which having been done in the CoMan model.

Further information and references

Robotran is based on a well-developed multibody mechanics theory, which needs some time to be digested. This report does not claim to cover it and the interested reader can find out more about the multi-body formalism and the underlying conventions, terminology and theory of the Robotran environment in the book [SF03]. To get started with Robotran, one can consult the official Robotran website (`www.robotran.be`) and the Robotran starting guide [CER09].

2.2.2 Matlab vs. Simulink

The Robotran framework is implemented both in Matlab and Simulink. The general overview presented in the previous section is valid for these two flavours of Robotran. This section is partially based on the formation given by the Robotran team at IIT in August 2012 [UCL12].

Matlab

The Matlab version of Robotran has the following advantages. First of all, it is written with the Matlab language and consists in a set of Matlab scripts and functions. This means that the environment is fully integrated in Matlab, which provides the user with all the facilities for visualizing and manipulating data. On top of that, the code is easy to debug, given all the debugging tools provided with Matlab (easy access to the variable contents, Profiler tool, etc.). Finally, all the Robotran analysis modules are available in the Matlab version of the framework, which is not the case with the Simulink version as we will see later.

However, the Matlab code is not compiled but rather interpreted. Consequently, it runs much slower than a compiled program can do. All in all, the Matlab version of Robotran is useful to design the first "prototype" of the model, given the debugging facilities available on Matlab and the flexibility of the interpreted Matlab language.

Simulink

Beside the Matlab version, there also exists a Simulink implementation of Robotran. This version is written in C and takes the form of a MEX-file[3] used inside a Simulink model. Therefore, the model is simulated using the Simulink integrator and must be interfaced with the Simulink simulation engine. Thus, the MEX-file is not a standalone executable file and Simulink is still needed to run the simulation.

However, the interfacing between the user model and the Simulink engine is performed by the Robotran framework and is thus transparent for the user. Indeed, the user files keep the same organization and nearly the same names as with the Matlab version. However, the Matlab code must be manually translated into C code by the user. This operation takes some time but is rather systematic and the Robotran user's guide gives some hints to perform a clean translation.

The development flow with Simulink is then nearly the same that the one with Matlab except that a few extra steps are required. The first one, as mentioned above, is the translation from Matlab to C. Then the C code must be compiled. Finally, the user must create a Simulink model to include the MEX-file and run the simulation.

Robotran with Simulink has the following advantages. First of all, the code is now compiled and not interpreted anymore. The simulation is thus far more efficient than with Matlab (it runs approximately a hundred times faster with our model). Moreover, the user has now the possibility to integrate the MBS model in a more general Simulink diagram. This can be useful to integrate the MBS model in a block-diagram based control scheme. This offers also the opportunity to perform multiphysics simulation or co-simulation by interfacing Simulink with other simulation softwares.

Nevertheless, there are also several drawbacks associated with the Simulink version of Robotran. The first point is that the C programming language is less flexible than the Matlab programming language. Indeed, the usage of C-code requires a more structured approach and the code must be compiled. Next, all the modeling possibilities are preserved with respect to the Matlab version but not all the analysis modules are available in Simulink. That is, only the direct and inverse dynamics analysis modules are available in Simulink[4]. However, these modules are the ones for which the simulation time may be critical and the other modules can be used within Matlab without any problem.

Finally, just as the Matlab version of Robotran is used to craft the prototype versions of the model, the Simulink version is used to design the final version. We followed the same steps for the development of the CoMan model (prototype on Matlab, final version on Simulink), as will be described in chapter 4.

To sum up, we had to use the Simulink version of Robotran in the design of the real-time walking controller because the version embedded in the CoMan could not be written in Matlab and also to achieve a reasonable simulation time to be able to perform optimizations with the model.

2.2.3 Pros and Cons

As explained in further details in [DMMC+12], the key advantages of the CoMan dynamic simulator developed with Robotran are the following.

- It generates efficient symbolic dynamical equations of the robot with high degrees of freedom.
- The user has the possibility to integrate custom models of the actuator dynamics (the passive elasticity and the DC motor equations), the ground contact model and fall detection.
- The models are generated in both Matlab and C languages. The user can then benefit from the respective advantages of Matlab and Simulink to prototype and design the model (as explained in the previous section).

However, Robotran is still in development and it is not entirely straightforward to make it run on Linux or MacOSX. Indeed, Robotran is mostly used on Windows at UCL and there are some peculiarities to cope with to make it work properly on other operating systems, especially for the Simulink version. But we managed to overcome these problems and are now able to use Robotran with the three aforementioned operating systems.

[3]MEX stands for Matlab EXecutable.

[4]The **direct dynamics** is the computation of the robot segments position given the actuator torques applied to the joints whereas the **inverse dynamics** consists in the computation of the joint torques given the robot segments position.

2.2.4 CoMan model in Robotran

As explained in section 2.2.1, the development flow of a Robotran model passes through different steps managed by different programs of the Robotran environment. The first one, MBsysPad, offers a graphical user interface, allowing the user to introduce the topology of the multibody system (MBS), as illustrated in figure 2.7. The MBS is described as an assembly of segments linked by joints (with potentially several DOF) and connected in a tree-like structure. This is represented in a 2D diagram but the actual model can, of course, be three-dimensional.

After the model simulation (in Matlab or Simulink), the results can be visualized with the Animation part of MBsysPad, as shown in figure 2.6. This program generates a video showing the dynamical behaviour of the MBS.

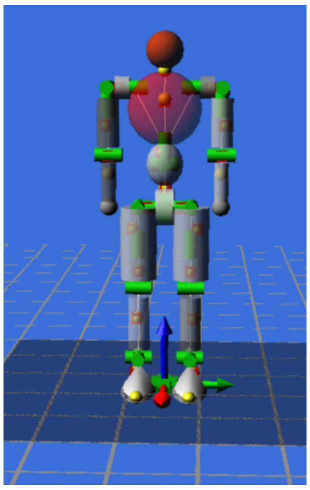

Figure 2.6: Visualization of the CoMan model in Robotran.

MBsysPad is written in Java, which ensures a good portability between different operating systems, and it output an XML file (<model_name>.mbs) containing all the model features. To find out more about this graphical model editor, the reader is invited to consult [CER09] and [DMMC$^+$12].

Beside MBsysPad, Robotran in also composed of MBsysTran, which generates the symbolic equations and MBsysLab, the simulator itself. The internal working of MBsysTran is beyond the scope of this report and we will not present it here. Simply speaking, it takes the MBS model and generates all the equations of movement symbolically.

Figure 2.8 shows the structure of the direct dynamics simulation environment of MBsysLab, i.e. the one used in our model. The user functions blocks are the ones where the user can include his own models. Here, the actuator dynamics is included in the files `user_JointForces` and `user_Derivatives` whereas the ground model in written in `user_ExtForces`.

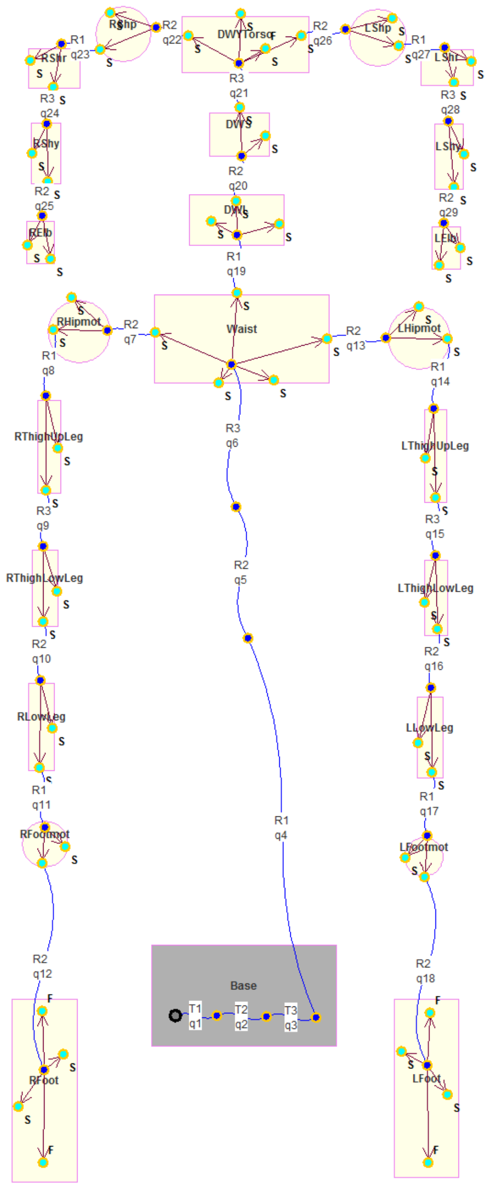

Figure 2.7: Floating base model of CoMan in MBsysPad (Robotran).

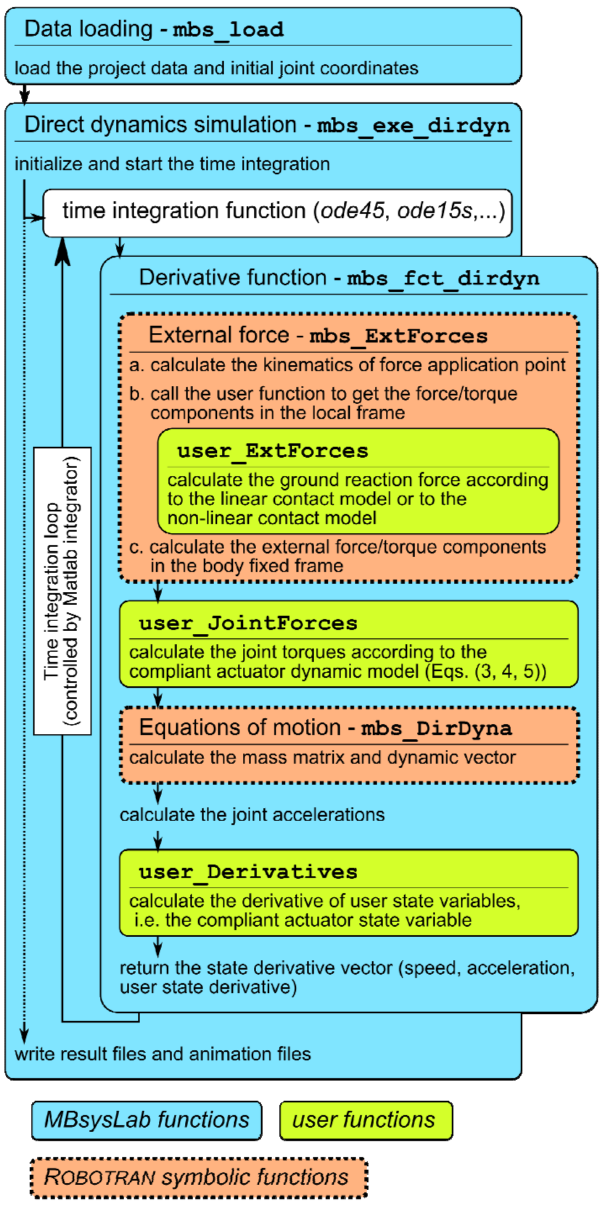
Data loading - mbs_load
load the project data and initial joint coordinates
Direct dynamics simulation - mbs_exe_dirdyn
initialize and start the time integration
time integration function (ode45, ode15s,...)
Time integration loop (controlled by Matlab integrator)
Derivative function - mbs_fct_dirdyn
External force - mbs_ExtForces
a. calculate the kinematics of force application point
b. call the user function to get the force/torque components in the local frame
user_ExtForces
calculate the ground reaction force according to the linear contact model or to the non-linear contact model
c. calculate the external force/torque components in the body fixed frame
user_JointForces
calculate the joint torques according to the compliant actuator dynamic model (Eqs. (3, 4, 5))
Equations of motion - mbs_DirDyna
calculate the mass matrix and dynamic vector
calculate the joint accelerations
user_Derivatives
calculate the derivative of user state variables, i.e. the compliant actuator state variable
return the state derivative vector (speed, acceleration, user state derivative)
write result files and animation files
MBsysLab functions
user functions
ROBOTRAN symbolic functions

Figure 2.8: Function diagram of the direct dynamics simulation environment of MBsysLab (from [DMMC+12]).

Chapter 3

Controller

The simulator is a powerful tool to implement a good controller. It is easier and less dangerous than implementing it from scratch on the real CoMan. Moreover, simulators allow us to constrain the robot while it might be difficult to apply these constrains on the real CoMan.

For instance, the model developed in this report is only a 2D walking controller: the simulator prevents the robot from falling on the left or on the right side. Developing a controller able to walk in an unconstrained environment (3D walking) will be studied next semester. This 2D walking controller, which is based on a biologically-inspired model, will be described in the first part of this chapter.

After the presentation of the controller itself, the second part addresses another important issue: the gait initiation. This topic is still far from being solved and will be studied more extensively during the next semester but we will see that by simply using the same controller as the one used for 2D walking, we can still get interesting results.

3.1 A biologically-inspired model

Implementing legged locomotion on a bipedal robot is a complex and widely studied problem. All the different approaches have their advantages and their drawbacks. Here, we use a bio-inspired control based on Hartmut Geyer's work [GH10]. The key principle is the following: human locomotion is controlled by muscle reflexes which encode principles of legged mechanics, so locomotion is seen as a chain of reflexes.

There are two main steps in order to implement such a controller. The first will be to integrate virtual muscles in the robot controller to simulate them. There are 7 muscle groups for each leg, which can be seen in figure 3.1. On the CoMan these muscles will be implemented as Hill-type muscles.

The second step is to control them thanks to stimulations sent to these virtual muscles. These stimulations will be computed with the reflexes model developed by Hartmut Geyer.

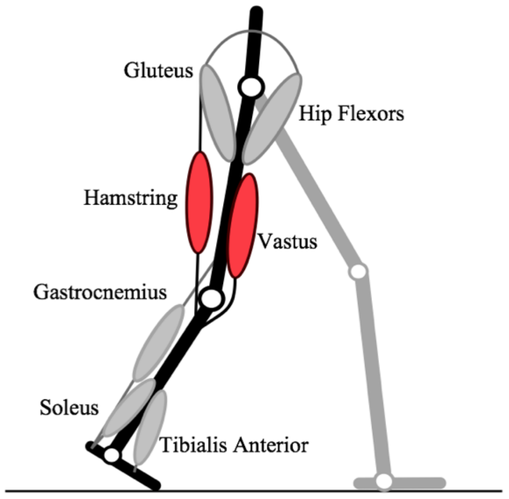

Figure 3.1: The 7 muscle groups of each leg

3.1.1 Virtual Hill-type muscles

Virtual muscles on the CoMan model

The muscle-reflex model developed by Hartmut Geyer represents the human body with a trunk and two three-segment legs. In this model, each leg is actuated by seven Hill-type muscles (see figure 3.1) that permit a direct comparison with prominent muscles of human leg [GH10].

In the real CoMan, each leg has three sagittal joints, two lateral joints and one yaw joint. By applying a zero-position controller on the non-sagittal joints, we get a leg similar to the one described in Hartmut Geyer's paper. If we apply a zero-position controller on all the joints of the upper part of the body, we also get something similar to the trunk described in Hartmut Geyer's paper (a trunk made of one piece with no additional degree of freedom). During next semester, we will investigate some possible improvements by actuating these upper-body part joints (swinging arms, help of the trunk joints to improve the stability for instance).

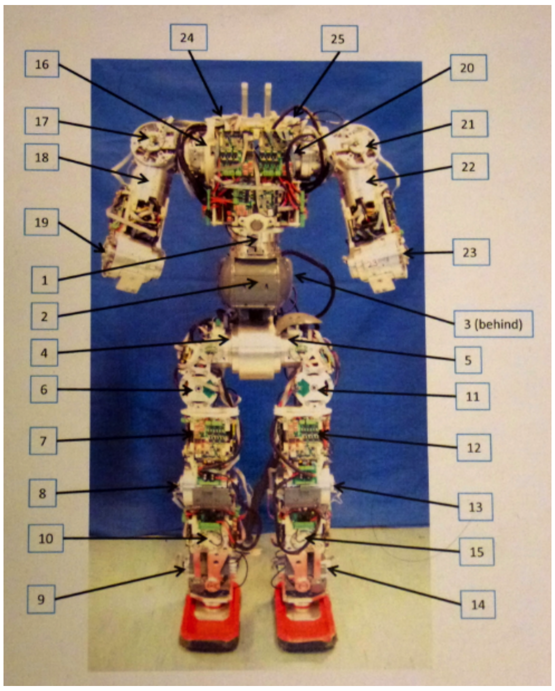

Figure 3.2: Sagittal joints: 4-8-9 (right leg) and 5-13-14 (left leg)

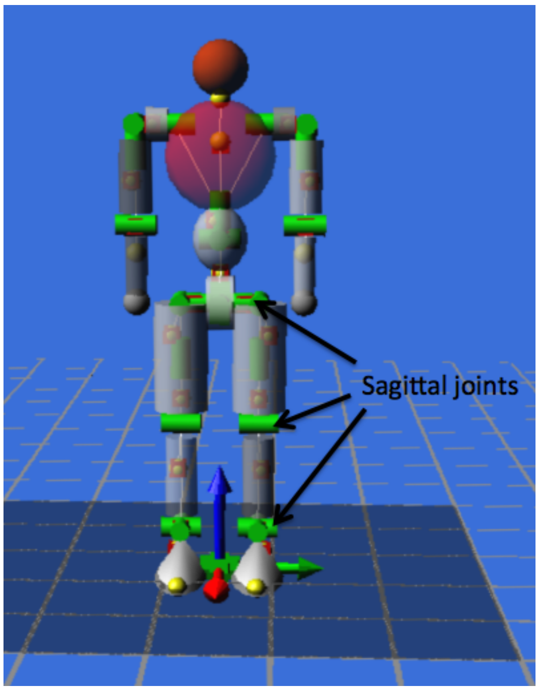

Figure 3.3: Sagittal joints in the simulation model (Robotran)

So, the real problem to fit Geyer's work are the seven muscles of each leg: to actuate the three sagittal joints of each leg, the CoMan uses DC motors located at each joint, which is very different from muscles contraction. Fortunately, Hartmut Geyer's model compute the torques generated by these virtual Hill-type muscles at each sagittal joint.

So, the strategy can be seen in figure 3.4: we start by computing the forces, lengths,... of virtual muscles as if there were attached to the CoMan legs. Then, it will be possible to derive the stimulations that control these muscles and to compute the forces they would apply as a consequence on the legs. Finally, these forces on the legs are converted into torques acting on the sagittal joints. These torques will be the references used in the torque controller of each sagittal joint of the legs.

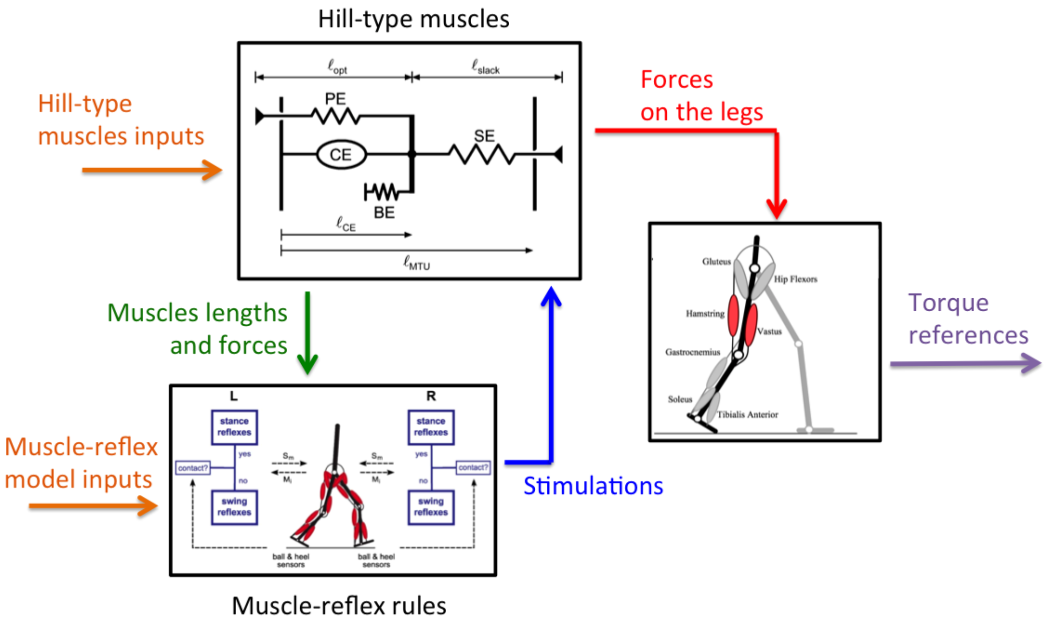

Figure 3.4: The two main parts of the controller are the virtual Hill-type muscles and the muscle-reflex rules. The final output of this controller are the torque references.

Hill-type muscles: presentation

Each leg is actuated by seven Hill-type muscles which are representations of the muscles mechanical response. For each of these muscles, we talk about a muscle-tendon unit (MTU) as we can see in figure 3.5. While we will not describe all the equations related to this representation of the muscle mechanical response, the key ideas will be presented in this report. All the equations implemented in this model can be found in [GSB03] and [GH10].

The MTU unit is composed of a contractile element (CE) and a series elastic element (SE). These form the MTU in normal operation.

A parallel elastic element PE is used when the CE stretches beyond its optimum length and a buffer elastic element BE prevents the active CE from collapsing when CE is under its minimal length.

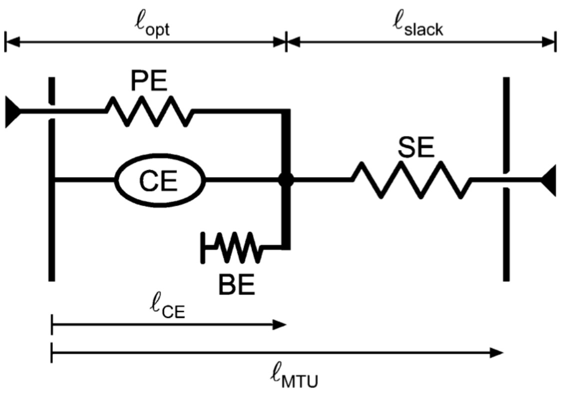

Figure 3.5: Muscle-tendon model (MTU)

If we assume that we know all the joint positions of the robot, only one state variable is needed for each MTU: the length of the contractile element l_{CE} (see figure 3.5). Indeed, the length of the MTU unit l_{MTU} can be computed from the joint angles and the length of the series elastic part l_{SE} can be computed from the equation $l_{SE} = l_{MTU} - l_{CE}$. So, knowing the state of all muscles is enough to compute the torque references. To understand how it works, we will now look at the different tools needed to get these references.

To get the torques applied on each sagittal joints of the legs, we need to find the forces F_m of all muscle-tendon units, where F_m is equal to the forces developed in the series elastic part SE: F_{se}. Each sagittal joint depends on three muscles: SOL, TA and GAS for the ankle, GAS, VAS and HAM for the knee and HAM, GLU and HFL for the hip. As we can see, two of these seven muscle groups have a contribution on two sagittal joints at the same time: GAS and HAM. All these muscles can be seen in figure 3.6. Knowing the forces applied by each MTU and knowing the anchor points of these MTUs on the leg segments is enough to compute the torques generated on each sagittal joint.

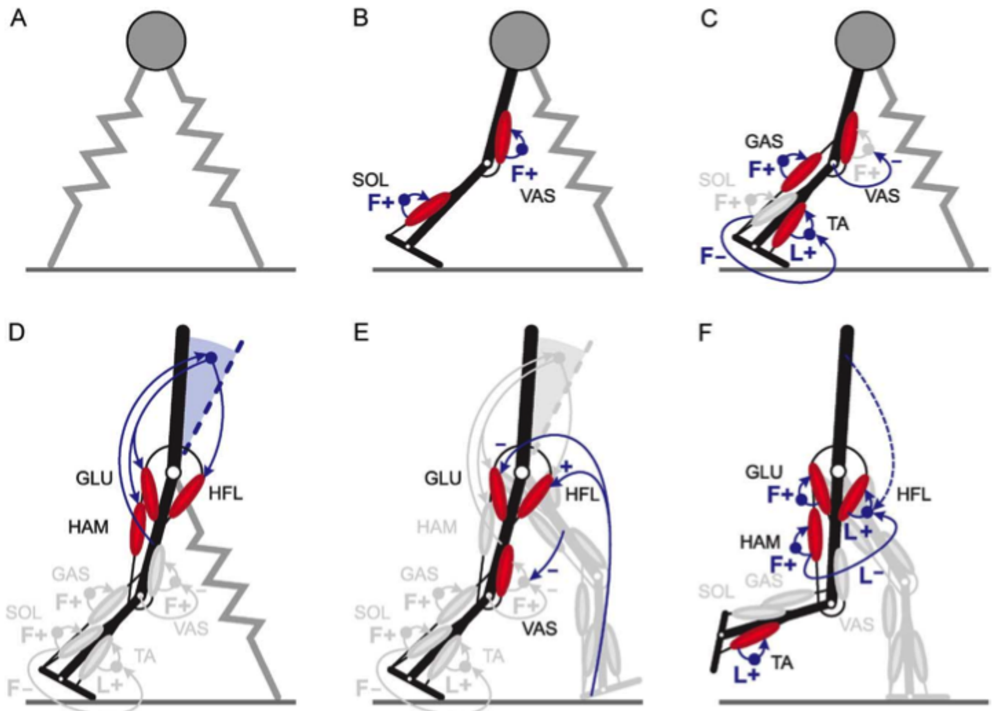

Figure 3.6: Muscles used in Hartmut Geyer's model and associated reflexes

So the only things missing to compute the torque references are the forces F_m applied by all the MTUs. These forces only depend on the lengths l_{SE} of the series elastic element in the MTUs. As we explained, we can compute l_{SE} from the state of the MTU ($l_{SE} = l_{MTU} - l_{CE}$). So, we will now look at the computation of this state.

Hill-type muscles: MTU state iteration

To get the state of each MTU, we use a numerical time integration: at each time step, we compute the new MTU state ($l_{CE,i}$) from the previous state ($l_{CE,i-1}$). We get $l_{CE,i}$ from the velocity v_{CE} of the contractile element: $l_{CE,i} = l_{CE,i-1} + v_{CE} \cdot \Delta t_i$ where Δt_i is the integration time step[1].

The tricky part in this problem is to get v_{CE}. To get a better understanding of how it works and how we can command these muscles with activation signals, we need to look at the computation procedure. First of all, we compute the forces acting on different parts of the MTU: F_{se}, F_{be} and F^*_{pe} which is immediate with the state of the MTU. Then, we derive $f_l(l_{CE})$ and $f_v(v_{CE})$ which are the force-length and the force-velocity relationships of the contractile element CE. By closely looking at these two equations ($f_l(l_{CE})$ and $f_v(v_{CE})$), we can see that they are monotonically increasing (f_l increases when l_{CE} increases and f_v increases when v_{CE} increases). More details are available in [GSB03].

[1]Even if it is possible to use the tools provided by Robotran to automatically compute this time integration, we will implement it ourselves in a discrete time controller in order to be able to transfer the controller code developed in simulation on the real CoMan.

The next step is to get the activation signals $A_m(t)$. These signals are the control signals of the MTUs and are related to their corresponding stimulations $S_m(t)$ we got with the muscle-reflex model. They are bounded between 0.01 (minimal activation) and 1 (maximal activation). More details on these will be given in section 3.1.2 (*Muscle-reflex model*). Currently, we will suppose that these activations are given. Equation (3.1) describes the equation used to compute f_v where F_{max} is a constant representing the maximal force this muscle can develop.

$$f_v(v_{CE}) = \frac{F_{se} + F_{be}}{A_m \cdot F_{max} \cdot f_l(l_{CE}) + F^*_{pe}} \tag{3.1}$$

When we want to contract a muscle, we send a higher activation A_m. As a consequence, f_v will decrease and v_{CE} will also decrease ($f_v(v_{CE})$ is monotonically increasing). When v_{CE} will be negative, the length of the contractile element CE will decrease and the muscle will contract. As l_{CE} decreases, $f_l(l_{CE})$ will decrease too ($f_l(l_{CE})$ is monotonically increasing) and will counteract the shortening of l_{CE} caused by the higher activation A_m. This will also have consequences on the other terms of equation (3.1). At the end, l_{CE} will converge towards a new constant length (if activations and joint positions do not change).

In figure 3.7 representing the lengths l_{CE} in the left leg, we send the minimal activation (0.01) to all the muscles. At $t = 2\,s$, only one activation signal is changed: there is a linear increase of the activation of the HFL muscle from 0.01 to 1.

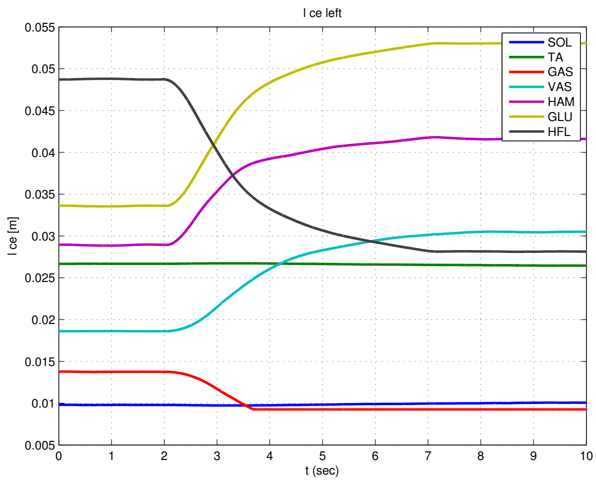

Figure 3.7: Lengths of the CE part of the Hill-type muscles (l_{CE}) of the left leg

Even if only the activation of the HFL muscle changed, this has also an impact on other muscles. We can see that the only two muscles whose lengths l_{CE} do not changed are TA and SOL, the muscles acting only on the foot. The reason is that activating the HFL muscle will flex the hip, which will move the knee. So the five muscles acting on these two joints will be impacted.

We can see that, after a few seconds, the new lengths of the contractile elements converge towards new constant lengths. The length l_{CE} of the HFL muscle is shorter because a higher activation means a higher contraction of the muscle.

To summarize, these are the different steps for each iteration:

- compute the muscle lengths l_{MTU} with the joint angles
- compute the forces F_{se}, F_{be} and F^*_{pe} based on l_{MTU} and on the previous state ($l_{CE,i-1}$)
- compute f_l
- compute the activation signals based on Hartmut Geyer's reflexes rules
- compute f_v with equation (3.1)
- compute v_{CE} with $f_v(v_{CE})$
- integrate v_{CE} to get $l_{CE,i} = l_{CE,i-1} + v_{CE} \cdot \Delta t_i$
- compute the torque references based on F_{se}

Hill-type muscles: MTU state initialization

As explained, there is only one state variable needed for each MTU: the length of the contractile element part: l_{CE}. We integrate the current velocity of this contractile element v_{CE} to get the new state: $l_{CE,i} = l_{CE,i-1} + v_{CE} \cdot \Delta t_i$.

However, there is still a problem: we do not know the initial values $l_{CE,0}$ of these lengths. Indeed, we only have some equations to estimate their order of magnitude. Actually, this is not a problem because taking a realistic solution $l_{CE,0}$ will converge to a stable value.

One important consequence of this for the implementation on the real CoMan is that it is non-sense to apply the torque references computed from this model before the muscle lengths converge. Fortunately, this system converges in less than 0.5 s most of the time.

Hill-type muscles: integration time step

The simulation model is computed using a numerical time integration with a constant integration time step. This time step is very important: if it is too high, high numerical errors will occur; if it is too low the simulation will be too slow (i.e. high simulation time). As we can see in figure 3.8, results are not good with a time step of 1 ms: this figure represents the forces on the seven muscles of the right leg which sometimes seem very shaky. This is not a natural behaviour, it is related to numerical integration problems (integration time step too high). So, when we take a smaller time step (0.5 ms), results are better (less shaky behaviour) as we can see in figure 3.9, but they are still not perfect. Furthermore, the simulation takes approximately twice the time needed before (because the integration time step is two times lower).

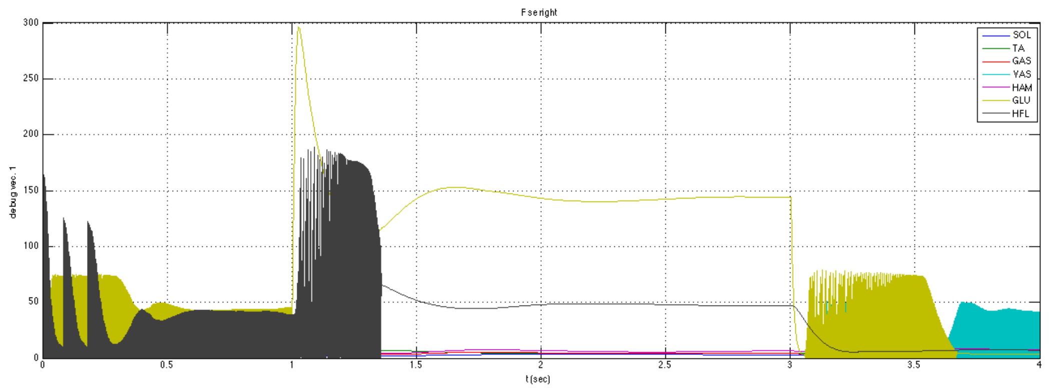

Figure 3.8: Forces on the seven muscles of the right leg. We can see that they sometimes seem a bit shaky, which is not a natural behaviour. This is related to numerical integration problems.
Integration time step: Robotran: 1 ms Controller: 1 ms

To solve this problem without having to reduce the integration time step, we have to understand what happens at each iteration: the controller computes the torque references and then Robotran uses this torque references to compute the dynamical model.

This integration time step problem is in fact related to the controller (so, to the Hill-type muscles model) and not to Robotran dynamic equations. Fortunately, the time needed for the controller is very small in comparison to the time needed by Robotran for the dynamics. Consequently, a simple solution is to use different time steps for Robotran (i.e. to compute the physics of the robot) and for the muscles model.

In figure 3.10, we take the same integration time step as in figure 3.8 (1 ms) for Robotran but we take a time step of 0.05 ms for the controller. Now, instead of having one iteration for the controller and then one iteration for Robotran, we have twenty iterations for the controller and then only one iteration for Robotran (but with a time step 20 times bigger). As we can see in figure 3.10 we have very good results (no shaky behavior) and, furthermore, the simulation is nearly as fast as the one related to figure 3.8.

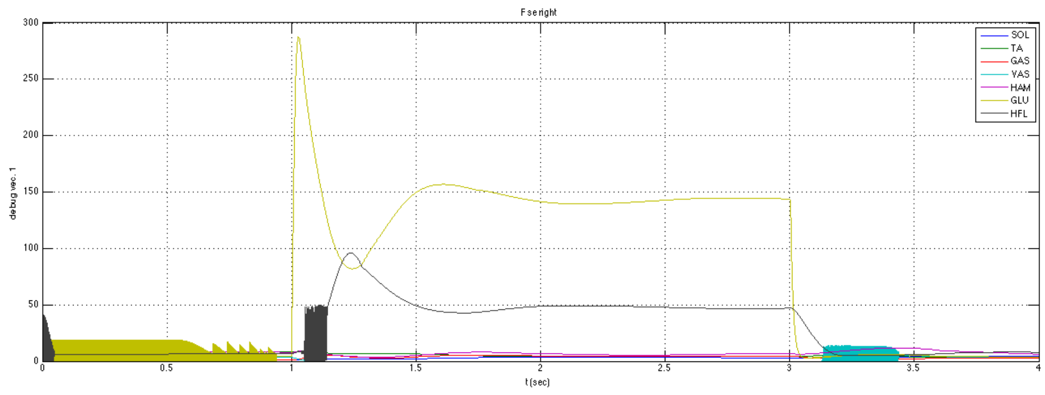

Figure 3.9: Forces on the seven muscles of the right leg in the same situation as the one described in figure 3.8. These forces seem less shaky, but we still have sometimes shaky behaviors.
Integration time step: Robotran: 0.5 ms Controller: 0.5 ms

One interesting thing to mention is that this integration time problem only occurs for small activations and depends on the model used. For instance, we do not have such catastrophic results as the ones from figure 3.8 when we apply this with the body parameters (muscle maximal forces, neutral lengths,. . .) of the model used by Hartmut Geyer. The reason is that the parameters are not the same for both models (Hartmut Geyer and the CoMan) as explained in section 6.1 where we address dynamic scaling.

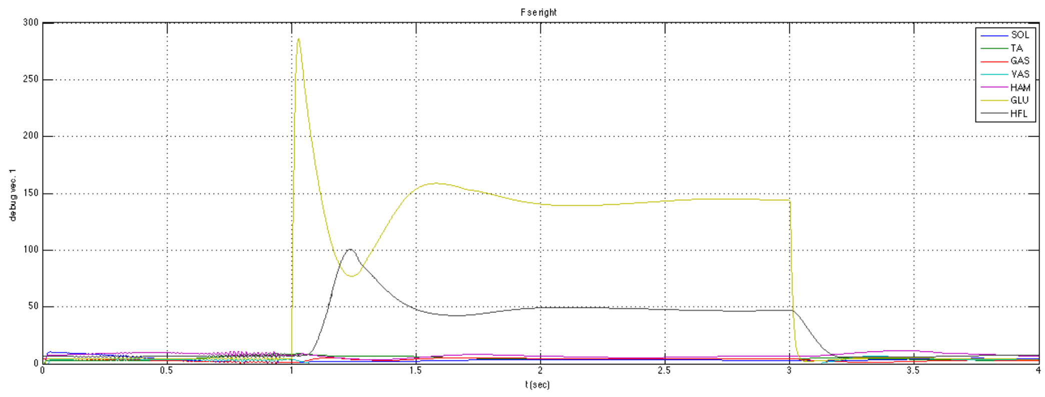

Figure 3.10: Forces on the seven muscles of the right leg in the same situation as the one described in figure 3.8 and 3.9. We do not have any shaky behavior.
Integration time step: Robotran: 1 ms Controller: 0.05 ms

Joint soft limits

As humans, we have limits in our joint ranges. For instance, we cannot do knee overextension. This is the same on the real CoMan but unfortunately, the model developed so far does not take this into account. To take this into account, joint soft limits are described in [GH10]. The idea is the following: apply an additional torque on each joint which stretches beyond its limits (so it is possible to slightly stretch beyond this limit, that is why this is called soft limit).

Unfortunately, this soft limit can vary very fast, and the torque controller of the CoMan will not be able to follow the torques references. To solve this problem, we apply a low-pass filter on the torques computed from the joint soft limit.

3.1.2 Muscle-reflex model

Once the virtual Hill-type muscles are implemented, we need to control them in order to allow the CoMan to walk. To control these muscles, we have to compute the stimulation $S_m(t)$ of each muscle which is a neural input related to the activation $A_m(t)$. This relation is described by a first order differential equation (3.2):

$$\tau \cdot \frac{d\,A_m(t)}{d\,t} = S_m(t) - A_m(t) \tag{3.2}$$

where τ is the excitation-contraction constant. These stimulations $S_m(t)$ will be sent to the virtual Hill-type muscles (see figure 3.4).

This section describes how these stimulations are computed with Hartmut Geyer's muscle-reflexes model.

Swing and stance

There are two major phases in the walk cycle for each leg: swing (when the leg does not touch the ground) and stance (when there is a contact with the ground), as we can see in figure 3.11. Depending on the current phase, we have different equations to compute the stimulation of each muscle. To detect in which phase we are, we use the feet ground sensors. If the force under one foot exceeds a fixed threshold, this means that the corresponding leg is in stance phase.

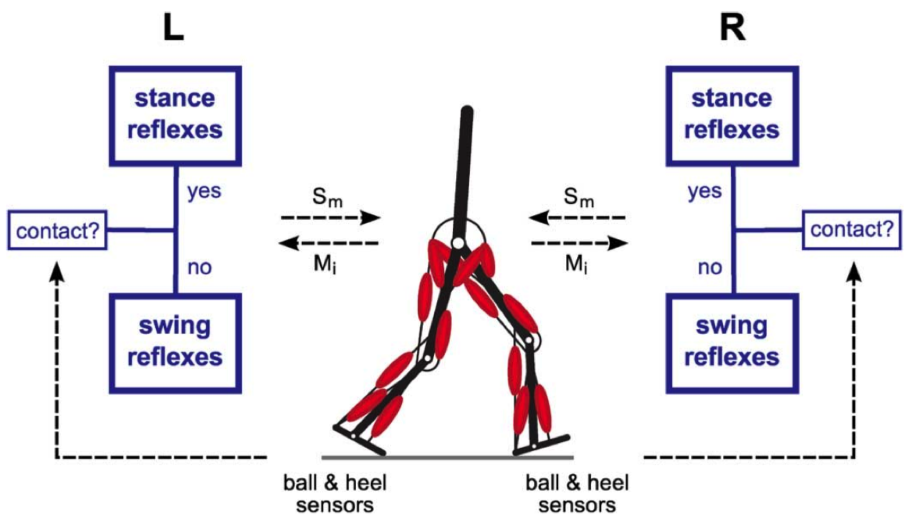

Figure 3.11: Selection of swing or stance reflexes based on the feet ground sensors.

Another important key feature is to detect if a leg is finishing the stance phase while being in double support (both legs on the ground, in stance phase). In this case, an extra contribution ΔS (a constant to tune) will be added to the stimulation of the three muscles acting on the hip joint (see [GH10] for more details). This term ΔS plays an important role in walk initiation, as we will see in section 3.2.

Parameters to optimize

To compute the stimulation signals, we use the muscle-reflex rules from Hartmut Geyer (see [GH10]). For instance, the stimulation S_{TA} of the TA muscle during the stance phase is computed with this rule:

$$S_{TA} = S_{0,TA} + G_{TA} \cdot (l_{CE,TA} - l_{TA}^{off}) - G_{SOL-TA} \cdot F_{SOL} \tag{3.3}$$

$l_{CE,TA}$ is the length of the contractile element of the TA muscle and F_{SOL} is the force applied by the SOL muscle (see section 3.1.1), but $S_{0,TA}$, G_{TA}, l_{TA}^{off} and G_{SOL-TA} are currently unknown. In fact there are 27 unknown parameters to find, which can be seen in table 3.1. Some of these parameters only act during swing phase, while others play a role only during stance phase, and finally, some of them are used in both phases. We suppose here that the two legs of the CoMan are identical and so are the gains for both legs.

These parameters will be found by optimization. At first, the model will use random parameters which will probably lead to the fall of the CoMan but, as time passes, the model will learn from its errors to progressively converge towards good parameters. This is similar to children learning to walk: at first they try to make a few steps and they fall. But they do not discourage and they learn themselves from their falls, as if they were trying to find the good parameters which will allow them to walk. So, like a real human, the CoMan must learn to walk.

Pre-stimulations	**Muscles gains**	**Offsets**	**Angles & feet forces gains**	**Special parameters**
$S_{0,SOL}$	G_{SOL}	l_{TA}^{off}	k_{ϕ}	ΔS
$S_{0,TA}$	G_{TA}	ϕ_{knee}^{off}	k_{bw}	θ_{ref}
$S_{0,GAS}$	G_{SOL-TA}	l_{HFL}^{off}	k_p	
$S_{0,VAS}$	G_{GAS}	l_{HAM}^{off}	k_d	
$S_{0,HAM}$	G_{VAS}		k_{lean}	
$S_{0,GLU}$	G_{HAM}			
$S_{0,HFL}$	G_{GLU}			
	G_{HFL}			
	$G_{HAM-HFL}$			
7 param.	*9 param.*	*4 param.*	*5 param.*	*2 param.*

Table 3.1: 27 parameters to optimize for Geyer's mucle-reflex rules

Among these 27 parameters, we have 7 pre-stimulation parameters which describe a constant value added to all the stimulations no matter what is the current situation of each leg. Then, we have 9 gains related to the lengths of the muscles and the forces applied by these muscles, then 4 offsets and then 5 gains related to the angle of the trunk, of the knee or related to the feet forces. The last two parameters to optimize are special parameters which play a very important role, especially in walk initiation as described in section 3.2.

All these 27 parameters are normalized. For instance, some of them are applied on length and muscles forces normalized by the optimal length or by the maximal forces. In [GH10], there are bounds for the search space of these 27 parameters. As they are normalized, we can keep the same bounds but we extend them a little bit to have a higher search space.

Noise on the stimulation signals

All the stimulation signals are bounded between 0.01 and 1 and result in a precise control of the different muscles. Nevertheless, there is noise on these signals for real humans. So, we also apply noise on these signals on the CoMan. In this noise implementation, the noise maximal magnitude is proportional to the stimulation (with a maximum of a 5% noise for the maximal stimulation). Another solution is to add noise on the activation signals which are the low-pass filtered stimulation signals. Nonetheless, we modify the noise signals only every 0.1 s, which is higher than the time constant of this low-pass filter. As a consequence, adding noise on the stimulation signals or on the activation signals is approximately the same.

Being closer to the biologically-inspired model is not the only reason to apply this noise. We want to get robust parameters at the end of the optimization, not only parameters that are dependent on one particular simulation environment. Otherwise, we will get very good results (in term of energy, . . .) but a little change in the environment (a very small obstacle, bad modeling of one aspect, . . .) will probably lead to the fall of the CoMan. This is the case when the solution has a very small basin of attraction: this solution might be very good when we are in this basin of attraction but a little perturbation might drive the system out of this basin of attraction, leading to the fall of the CoMan. Using a noise stimulation in the optimization will solve this problem by choosing maybe less efficient parameters but more robust ones (i.e. with a larger basin of attraction).

Moreover, by applying a noise proportional to the stimulation level, these robust parameters might be in fact more efficient. The reason is that this strategy may promote solution levels with lower stimulations (because they will be less sensitive to noise).

3.2 Walk initiation

3.2.1 Geyer's model alone is enough to initiate the gait

Hartmut Geyer's muscle-reflex model is used during walking to generate a cyclic locomotion with a chain of reflexes. One important question is to know if this model can be used for walk initiation. This is an interesting topic as we need to implement this algorithm on the real CoMan. Consequently, it would be difficult to give it a precise initial speed and initial position as was for instance done in [Ber11].

First, we tried to initiate the walk by applying a position controller on the sagittal joints and to launch Geyer's model at the same time. After a short time (0.6 s), we stopped the position controller and let the CoMan walk with only the torque control from Geyer's model on the 6 sagittal joints of the legs. We already achieved some interesting results with this method.

Nonetheless, we could see that it is possible to initiate the walk without using any position controller, just by letting the optimization find good parameters (for Geyer's model) when the CoMan starts motionless upright, i.e. in home position.

3.2.2 First leg in swing phase

Instead of a position controller which imposes which will be the first leg in swing phase, the situation is now symmetric for both legs at the beginning. So, we do not encode in the algorithm which will be the first leg in swing phase.

One surprising fact is that the model can choose between the two legs which one will be the first one in swing phase while the other remains in stance phase. Moreover, the first leg in swing phase can be different from one trial to another (without changing the algorithm) depending on the noise on the signals.

This idea can be understood by looking at one of the last parameters from Hartmut Geyer's model: ΔS (see table 3.1). This term is applied to the muscle groups GLU and HFL (two MTUs whose effects apply on the hip joint) when both legs are in *stance* phase (double support) and when the corresponding leg is in *finishing stance* phase. The optimization found a very high value for this parameter ΔS. As a consequence, it has a major impact on the hip torque. To know which leg is in *finishing stance* phase and will receive this extra term ΔS, we look at the forces under each foot (see figure 3.11). The leg in *finishing stance* phase will be the one with the lower ground reaction force under the foot.

Thanks to this, we can understand what is happening: a slight difference perceived in the feet forces will be enough to select which leg is in *finishing stance* phase and will receive an additional term ΔS. This term will help this leg activating the hip muscles to start walking. After a short time, this leg will not touch the ground anymore and will enter in *swing phase*. Thanks to this, the CoMan will enter the cyclic locomotion.

This fact was observed in simulation and on the real CoMan: without changing the algorithm, the CoMan changes the first leg in *swing phase* depending on the trial (because of the noise introduced in the model for the simulation; on the real CoMan, there is already natural noise on the feet force sensors). These results will be described in section 7.2.1.

This fact can be compared to what happens with humans: when we start walking, we do not think at which will be the first leg in *swing phase*. Most of the time, it is the same because we have a leg a bit stronger than the other but it may happen that we start by moving the other leg, without having to think about it (for instance when we are on an uneven terrain).

Another important fact related to this walk initiation is that the CoMan is very sensitive to the reference angle θ_{ref} of the trunk (with respect to the direction of the gravity vector), which is the last one of the 27 parameters we optimize (see table 3.1). The reason is that this angle is also related to the muscles acting on the hip joint (as for the term ΔS). The walk initiation is very sensitive to these muscle stimulations, which explains why this parameter is so important for walk initiation.

Chapter 4

Model Design

After the controller design, we had to integrate it on a model of the dynamic behaviour of the robot. The first version of this model, that we received from IIT at the beginning of this project, was implemented only in Matlab. As explained in chapter 2, we had thus to develop it further and port it in Simulink to be able to run fast simulations. This chapter sums up the main steps in this robot model development.

4.1 Overview & Perspectives

4.1.1 Model development

In this project, we developed not only the Geyer's muscle-reflex model but also the robot dynamic simulator, which computes the behaviour of the CoMan seen as a multibody system. This dynamic simulator is a physical model of the CoMan on its own and can be used for other applications than just testing Geyer's controller.

The current model structure is the following:

1. **External layer:** Simulink engine to run the simulation (integrator, I/O interface from/to Matlab);
2. **Robotran layer:** Robotran framework files (part of the MBsysLab environment);
3. **Controller layer:** Hill-Geyer's model code, which can be used both in the simulator and in the real CoMan.

The key advantage of our approach is that it is possible to reuse the exact same code for the controller both in simulation and with the real CoMan. We achieved this by keeping a neat separation between the simulator (which computes the physics of the robot) and the controller (Hill-Geyer's model), with clear interface between the different elements. To do so, we had to develop the whole model in C-code and we used Robotran as an underlying framework to develop our model (this point will be detailed in section 6.3).

We used the Robotran framework as a base to develop our model. We use Matlab and Robotran to visualize and process the data generated by the simulation, thanks to the graphical tools provided with these two programs.

The general approach followed in the development of the model is the following:

1. IIT model (Matlab);
2. Cleaned model, using Robotran standard functions (Matlab);
3. Hill & Geyer's model (Matlab);
4. Hill & Geyer's model (Simulink).

The development history is detailed in the next section (4.2).

4.1.2 Standalone version

The next step in the model development is to get rid of Matlab/Simulink and have a standalone version of the model. The main reason to do so is that Matlab is a heavy program and it is thus not practical to run lots of simulations in parallel. Indeed, on the optimization cluster we use to run the optimization (see chapter 5), we face some troubles related to the use of Matlab which must be running in background on each cluster core to simulate our model. Running so many instances of Matlab out of one single license seems to be a problem and we reached the conclusion that the optimizations would run much faster if we could get rid of Matlab/Simulink.

This evolution can be considered as the third major version of the model. First, we escaped from Matlab (slow due to the interpreted code) to go to Simulink (fast thanks to the compiled code). Now, the idea is to escape from Simulink to go to a standalone C-code version of the model. The history of the different versions (including the third version to come) is presented below:

- **Version 0.x** = CoMan model received from IIT, in Matlab (base version to develop the model);
- **Version 1.x** = CoMan model with Hill-Geyer's model implemented in Matlab (= *Matlab version*);
- **Version 2.x** = CoMan model with Hill-Geyer's model implemented in Simulink (= *Simulink version*);
- **Version 3.x** = CoMan model with Hill-Geyer's model implemented in C-code only (= *standalone version*).

4.1.3 Robotran with Simulink user's guide

To implement the CoMan model on Simulink from the Matlab version, we had to understand the internal working of the Simulink version of Robotran, which is slightly different from the Matlab version. This leads to the redaction of a user's guide [Bar10] (written in French) explaining all the peculiarities related to the translation of a Robotran model from Matlab to Simulink and how to develop a Robotran model with Simulink.

4.2 Development history

Step 1 – Starting version
We received the first version of the model from IIT. This model is for the 23-DOF version of the CoMan (i.e. the final one) but does not include the mass of the PC104 nor the one of the battery. Further more, even though this version is based on the Robotran functions (in Matlab), it is a custom-made script which does not use the standard Robotran function, but rather use the line of code that are stricly needed to run the simulation. Therefore, it was e.g. not possible to freeze easily the non-sagittal DOF of the robot, a mandatory step in the 2-D Geyer's controller design.

Step 2 – Model cleaning
In order to allow an easier manipulation of the model (e.g. freezing the non-sagittal DOF directly from MBsysPad) and also to be able to translate it easily to Simulink, we cleaned the model and managed to use the Robotran standard functions again. This was not entirely straightforward because some features, such as stopping the simulation when the robot falls, are not possible using the standard Robotran functions. Actually, this is the reason why IIT decided not to use the standard Robotran functions. Anyway, by modifying some of the Robotran core files, we managed to overcome this difficulty and implemented the same functionalities as present in the first version of the model but using Robotran standard functions.

Step 3 – Geyer's model implementation and test with Matlab
We implemented the Hill's and Geyer's models in Matlab language and begun to validate the Hill's model, i.e. hanging the CoMan in the air and activating one muscle after the other. Due to the very long computation time required to simulate the model in Matlab, we decided to not validate Geyer's model with this version, but rather wait for the one with Simulink.

Step 4 – Model translation to Simulink
Having a working model on Matlab and using the standard Robotran functions, we were now able to translate it to Simulink, i.e. translate the Matlab-code to C-code. We had to understand the – slightly different – code organization in Simulink and this lead to the writing of the aforementioned "Robotran with Simulink user's guide".

Step 5 – First tests with Simulink
After the translation of the code, we tested the Simulink model against the Matlab version. The results were the same as those given by Matlab and the computation time was much faster, as expected. This was a first functional version of the Geyer's model on Simulink.

Then, we re-validate the Hill's model in the same way as for the Matlab version. On top of that, we also tested the Geyer's model, i.e. the activations computed via the reflex rules. From this step we focused only on the Simulink model and we did not go back to the Matlab version. Nevertheless, we still use Matlab to analyse the data generated by the simulation.

Step 6 – Successive improvements of the Simulink model
We did some improvements in the model. First, we used a fixed time step integrator. Using this is really easy to do in Simulink but not in the Matlab version of the model because this is not included in the standard Robotran functions. Moreover, we changed the ground model to use the exact same model as suggested by Hartmut Geyer is his paper [GH10]. Finally, we implemented some functions to be able to begin the simulation with an arbitrary state of the robot (and not only in home position). These two last features are detailed in the next section.

Step 7 – Birth of GroundWalker and SkyWalker
Given the fact that the controller was now tested in simulation, we had to transfer it into the real CoMan. We wanted to use the exact same files for the controller in simulation and in the real robot. Therefore, we had to rewrite the model to draw clearly the frontier between the dynamic model of the robot (simulating the physics) and the controller.

This re-writing of the model gave birth to two different versions. The first one is called GroundWalker and is based on the previous model re-written in the way described just above. The second one is called SkyWalker and this is a version where the robot is hang up in the air attached by two points in the neck, as described in section 7.1. The name comes from the fact that the robot seems to walk in the air (and from a famous movie, too).

GroundWalker is thus the version used to study the walking on the ground, i.e. validate the Geyer's model, whereas SkyWalker is used to validate the Hill model in the air, i.e. without touching the ground.

From this step, we were able to launch an optimization on the optimization framework (more details in chapter 5) and we got a set of controller parameter allowing the robot to walk dynamically in a robust way.

Step 8 – Improving GroundWalker
As will be described in section 6.2 and as shown in figure 6.10, there is a 2 cm horizontal shift between the trunk and the legs axis. In Geyer's model, however, the trunk is aligned with the legs. Therefore, we had to modify the GroundWalker model to take this shift into account. The modifications have been done in two ways. First, we modified the way the trunk inclination is measured in the model to get closer to reality. Then, we corrected the measure to get the same result as we would have with the trunk aligned with the legs.

4.3 Important features

4.3.1 Ground contact model

We integrated the ground contact model (GCM) developed by J.-F. Collard (CEREM, UCL) in our CoMan GroundWalker model. This GCM is the exact same model as described in Hartmut Geyer's paper ([GH10], Appendix IV). To implement the GCM in GroundWalker, the Robotran model had to be thoroughly modified. Indeed, the foot design has been adapted to the one in Geyer's paper.

The GCM includes static and dynamic friction and a non linear reaction force from the ground. It is a 2D GCM, which computes only the reaction force normal to the ground and the one in the direction of motion of the robot. So, there is no lateral force generated by this model. To keep things as simple as possible, we locked all the non-sagittal dof of the CoMan's legs in order to be as close as possible to the situation described in Geyer's paper. A future work will obviously be to add a lateral horizontal reaction force and we will take care of this when we will study the 3D walking.

Table 4.1 contains the parameters of the ground contact model. We modified some of these parameters to have a more realistic behaviour of the foot on the ground. Indeed, with the parameter values given by Geyer, the ground seems very slippy. To avoid that, we increased the v_{gxMax} and v_{gzMax} parameters in order to have a more elastic impact, since it is explained in [GH10] that $v_{Max} = \infty$ gives to a perfectly elastic behaviour whereas $v_{Max} = 0$ describes a perfectly inelastic behaviour. Furthermore, we increased the horizontal stiffness of the ground k_{gx} to prevent the foot from slipping too far from its impact point.

Parameter	**Value in the CoMan model**	**Value in Geyer's model**
Stiffness of vertical ground interaction, k_{gz}, $[N/m]$	27818	27818
Max relaxation speed of vertical ground interaction, v_{gzMax}, $[m/s]$	0.1	0.03
Stiffness of horizontal ground stiction, k_{gx}, $[N/m]$	5000	2782
Max relaxation speed of horizontal ground stiction, v_{gxMax}, $[m/s]$	0.1	0.03
Stiction to sliding transition coefficient, μ_{stick}, $[-]$	0.9	0.9
Sliding friction coefficient, μ_{slide}, $[-]$	0.8	0.8
Sliding to stiction transition velocity limit, $vLimit$, $[m/s]$	0.01	0.01

Table 4.1: Ground contact model parameters

4.3.2 Starting from a given state

We wrote functions to make it possible to launch the simulation with the CoMan already walking and not in its home position. This is a necessary condition to be able to find a set of model parameters which are optimized to make the robot walk and not to make it start from home position and then walk. We must still decide exactly how much information from the previous state we keep in the initial state, but modifying the initial state is now easy to do, thanks to a set of scripts to extract, monitor and load the state in the CoMan.

Chapter 5

Optimization

As explained in section 3.1.2, there are 27 open parameters in Geyer's model, which must be carefully tuned in order to allow the robot to walk. To find a good tuning of these parameters (corresponding to a robust walking gait), it is necessary to go through an optimization step. We use the Particle Swarm Optimization (PSO) algorithm to perform this optimization. This algorithm has been implemented on the Optimization Framework developed by Jesse van den Kieboom.

Furthermore, the optimization might also be useful next semester to tune other parameters in order to allow the CoMan to walk in 3D or to combine CPGs and reflexes together, for instance.

5.1 PSO

This algorithm is inspired from bird flocking. It is thoroughly described in [KE95] and [CK02]. This is a cooperative, population-based optimization method. The parameters to optimize are represented by a set of particles (the population). Each particle lives in a N-dimensional space, where N is the number of parameters to optimize.

The optimization state is composed of all the positions (x_{ij}) and speeds (v_{ij}) of the particles. The position of a particle is its coordinates in the N-dimensional space of the parameters to optimize. Initially, the particles are randomly spread over the whole search space, i.e. the space of all possible values for the parameters. Then, a single-valued fitness function is computed for each particle (see section 5.2 for the details about the fitness function). Given the fitness value associated with each particle, their velocity is computed (equation 5.1) and then used to update their position in the search space (equation 5.4). After the position update, the fitness function is computed again for each particle, then each particle position is updated, and so on for a given number of iterations.

As detailed in the Optimization Framework manual [vdK11], the algorithm can be described by the following equations:

$$v_{ij}(t+1) = \chi\left[v_{ij}(t) + R_1\varphi_1\left(P_{ij} - x_{ij}(t)\right) + R_2\varphi_2\left(P_b - x_{ij}(t)\right)\right] \tag{5.1}$$

$$\chi = \frac{2k}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \tag{5.2}$$

$$\varphi = \varphi_1 + \varphi_2 > 4 \tag{5.3}$$

$$x_{ij}(t) = x_{ij}(t-1) + v_{ij}(t) \tag{5.4}$$

where i is the dimension index and j is the particle index.

The algorithm parameters are the following:

- χ is the constriction factor, determining the maximal velocity of the particles
- P_{ij} is the position with the best fitness in the particle history
- P_b is the position with the best fitness in the population history

- φ_1 is the cognitive factor, giving the relative importance to the particle history
- φ_2 is the social factor, giving the relative importance to the population history
- R_1 and R_2 are two pseudo-random number in the range $[0, 1]$

5.2 Fitness function

5.2.1 Overview

The fitness function is the metric to compute how good is a given solution. This is one of the most important aspect of the optimization because using one kind of fitness function instead of another will lead to very different results.

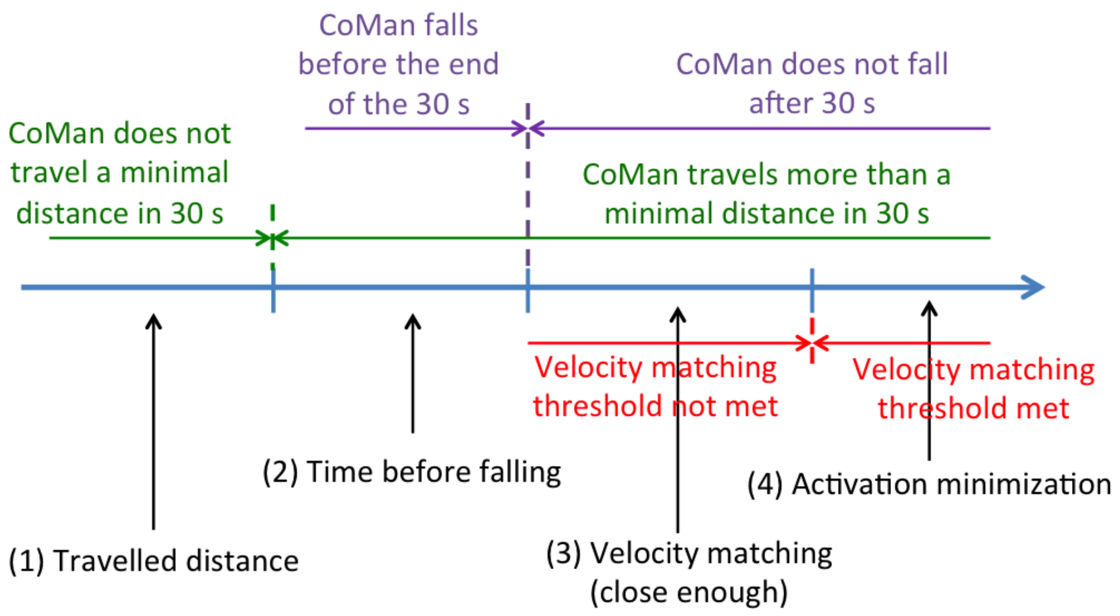
CoMan falls before the end of the 30 s
CoMan does not fall after 30 s
CoMan does not travel a minimal distance in 30 s
CoMan travels more than a minimal distance in 30 s
Velocity matching threshold not met
Velocity matching threshold met
(2) Time before falling
(4) Activation minimization
(1) Travelled distance
(3) Velocity matching (close enough)

Figure 5.1: Fitness function in 4 stages.

Figure 5.1 gives a overview of the fitness function we use to optimize the controller. It is composed of 4 stages:

- **Stage 1:** the CoMan walks on a minimal distance
- **Stage 2:** the CoMan does not fall during the simulation time
- **Stage 3:** the CoMan walks at a precise speed
- **Stage 4:** the total activation of all muscles is as small as possible

Each of these four stages is associated with a fitness (noted F) bounded between 0 and 100 with the following rules for the different stages:

- **Stage 1:** F is proportional to the fraction of the minimal distance travelled (we do not care if the CoMan travels more than the minimal distance)
- **Stage 2:** F is proportional to the time the CoMan walks before falling
- **Stage 3:** F increases when the square of the difference between the mean speed and the target speed decreases
- **Stage 4:** F increases when the total activation decreases

The idea behind this 4-stage function is that, at first, we only get the fitness of the first stage. When this stage is fulfilled, i.e. when the CoMan walks more than the minimal distance), we add the fitness of the second stage, and so on for the next stages. So, we add the fitness of the third stage when the CoMan does not fall during the simulation time and we add the fitness of the last stage when the current speed is close enough to the target speed.

5.2.2 Mathematical form

For the 3rd and 4th stages, where we need a fitness that decreases when the measured value increases, we use a mathematical function of the form

$$f(x) = \frac{100}{1 + k \cdot x}$$

where $f(x)$ is the fitness, x the measured value and k a parameter that controls the slope of the function $f(x)$. Thanks to this, we have $f(0) = 100$ and $f(+\infty) = 0$.

The fitness described in the previous section can also be expressed in mathematical form:

$$F = \underbrace{\left(\texttt{travelledDistance} \cdot \frac{100}{\texttt{minimalDistance}}\right)}_{\texttt{stage 1}} + \underbrace{\left(\texttt{currentSimulationTime} \cdot \frac{100}{\texttt{maxSimulationTime}}\right)}_{\texttt{stage 2}}$$
$$+ \underbrace{\left(\frac{100}{1 + \texttt{k}_{\texttt{velocity}} \cdot (\texttt{meanVelocity} - \texttt{refVelocity})^2}\right)}_{\texttt{stage 3}} + \underbrace{\left(\frac{100}{1 + \texttt{k}_{\texttt{activation}} \cdot \texttt{totalActivation}}\right)}_{\texttt{stage 4}}.$$

`totalActivation` is the sum of all activation signals at each time step, given by the following expression:

$$\texttt{totalActivation} = \sum_{t=0}^{t_{max}} \sum_{i} \left[(\texttt{a}[i](t) - \texttt{aMin})^2 \right]$$

where the activation is denoted $\texttt{a}[i](t)$ for the activation of one muscle, where `aMin` is the minimal activation (which we do not want to integrate) and where t_{max} is the total simulation time. The sum is performed for all muscles of both legs.

Each stage taken individually is bounded between 0 and 100. We only add the next stage when the previous one reached a value of 100, as explained before. The fitness is evaluated only at the end of the simulation.

5.2.3 Utility of the first stage

The first stage of the fitness function forces the robot to walk a minimal distance. While it has not walked that distance, the fitness cannot reach the second stage, i.e. it cannot exceed 100.

Initially, this stage was not included in the fitness. Then, the optimization converged to a strange result: the robot made one really big step and then remained locked, but it did not fall. He was in equilibrium in this configuration, so the first stage, which was then "minimum time before falling" was fulfilled, leading the optimizer to converge towards this solution. To avoid this, we added the current first stage to force the robot to walk a minimal distance.

5.2.4 Improving the fitness

This fitness function will probably evolve during the next semester. For instance, the activation minimization will probably be replaced by the minimization of the energy used by the muscles. This should not be difficult to implement, since we already know the force that the contractile elements apply and we also know the velocity of this contractile element part.

5.3 Results

We run several optimizations, improving each time the fitness or the robot model. Finally, we managed to get a stable gait in simulation, which was also robust to small perturbations. The results of that optimization are presented in this section.

5.3.1 Optimization & model parameters

In the optimization we use 100 iterations and 60 particles. Each particle considers the whole set of particles as its neighbours.

As explained in section 3.1.2, we add noise on the stimulation signals. To get parameters as robust as possible, we test the fitness function 5 times with the same parameters and we keep the worst fitness function among the 5. Consequently, if a given set of parameters makes the robot walk only once out of the 5 evaluation of the model, it is not taken into account in the resulting fitness function value.

The parameters used in this optimization are listed in table 5.1. The parameters used for the ground contact model are presented in section 4.3.1.

Parameter names	**Parameter values**
Simulation time	15 s
Integration time step	0.5 ms
Reference velocity	0.4 m/s
Minimal distance to walk	0.6 m
Threshold for fall detection	0.35 m
Velocity margin	0.05 m/s
Noise frequency	10 Hz
Noise amplitude	0.05

Table 5.1: Model parameters

Here are a few comments about some of the optimization parameters:

- **Threshold for fall detection:** when the waist of the robot goes below this threshold, it means that it has fallen and the simulation is stopped.
- **Velocity margin:** it established the threshold to allow the fitness to reach the 4th stage; if the difference between the mean velocity and the reference velocity is smaller than this parameter, then the 4th stage is accessible.

5.3.2 Discussion

Parameters for a 2-dimensional controller

Here, the robot starts from home position, as will be the case with the tests we run (cf. chapter 7). Therefore, the optimizer searches for a solution (a set of parameters) which allows the robot to initiate the gait and to maintain it robustly at the same time. This is not ideal. Indeed, a better solution would be to optimize only for parameters giving a stable gait and to not care about the initiation.

However, when running the tests on the actual robot, we always begin in home position, that is why we begun to run the optimization starting from this state. Then, we did not have enough time to run more elaborated optimizations. Nevertheless, this result is enough for the moment. Indeed, we still have a robust gait in simulation, starting from home position but allowing the robot to walk without falling. Moreover, the simulation is run for the 2-dimensional model of the robot. But in reality, we have no way to constraint the robot to move only in the sagittal plane. So, we have to maintain it by hand, which induces large perturbations. To sum up, given that the controller is still developed in 2-D and that the robot moves in 3-D, the parameters found after the optimization will probably need to be adapted. Therefore, there is no point in searching too far the optimal parameters for the 2-D controller and we focused on the first tests run on the robot with the parameters found after this optimization.

Fitness evolution & Geyer's parameters

Figure 5.2 shows the fitness progression during the optimization process (100 iterations of the PSO algorithm). We recall here that for each iteration the model is evaluated 5 times and the resulting fitness is the worst of the 5.

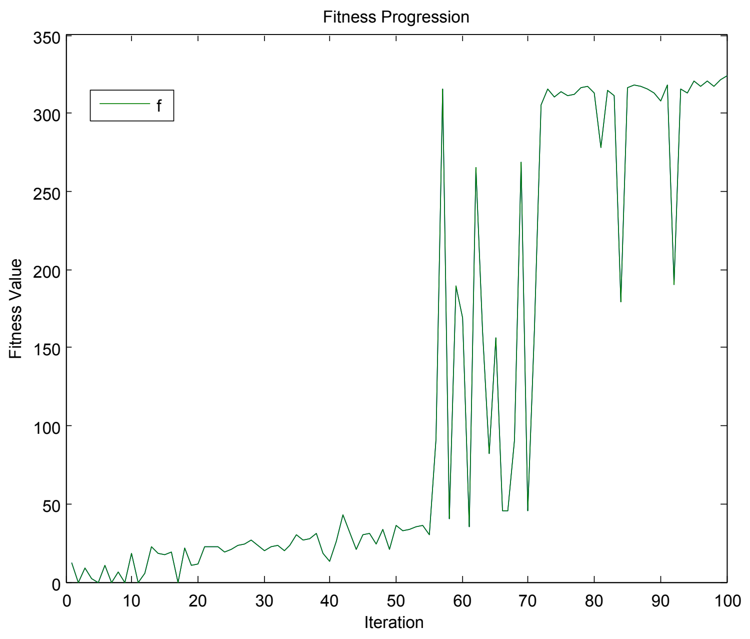

Figure 5.2: Fitness progression during the optimization: the fitness value is the one corresponding to the best solution obtained at a given iteration.

We can see that for the 50 first iterations, the model does not pass the first stage of the fitness, i.e. it is not able to walk robustly the minimal distance (the model has probably already walked the minimal distance but never 5 times in a row). After that, there is an intermediate step, between the 50th and the 70th iteration approximately, where sometimes there is a good solution and sometimes not. Indeed, when the optimizer finds a stable gait, there is a huge increase in the fitness value (the constraints related to the 2nd and the 3rd stage are then easy to match). Finally, after the 70th iteration, the model reaches the 4th stage of the fitness and stabilizes there. We get the best solution after the 100th iteration. The parameters values are given in table 5.2.

	value	min	max		value	min	max
$S_{0,SOL}$	0.0133	0.01	0.02	G_{HFL}	0.6969	0.1	4.0
$S_{0,TA}$	0.0129	0.01	0.02	$G_{HAM-HFL}$	85.8509	0.0	150.0
$S_{0,GAS}$	0.0145	0.01	0.02	l_{TA}^{off}	0.5227	0.5	0.95
$S_{0,VAS}$	0.2692	0.03	0.95	ϕ_{knee}^{off}	2.8271	2.5	3.14
$S_{0,HAM}$	0.0299	0.01	0.5	l_{HFL}^{off}	0.4414	0.0	0.8
$S_{0,GLU}$	0.0408	0.01	0.5	l_{HAM}^{off}	1.7233	0.75	2.0
$S_{0,HFL}$	0.4058	0.01	0.5	k_{ϕ}	1.5097	0.0	5.0
G_{SOL}	0.7345	0.7	3.0	k_{bw}	2.5897	0.05	6.0
G_{TA}	2.1304	0.4	4.0	k_p	5.4631	1.4	24.0
G_{SOL-TA}	0.0608	0.0	0.6	k_d	0.6500	0.05	1.0
G_{GAS}	0.6655	0.0	2.0	k_{lean}	2.6112	0.5	6.5
G_{VAS}	1.9257	0.6	15.0	ΔS	0.8862	0.1	1.4
G_{HAM}	0.6357	0.0	0.9	θ_{ref}	0.0492	0.017	0.11
G_{GLU}	0.3454	0.0	0.7				

Table 5.2: Geyer's model parameters values after optimization (corresponding to the best fitness, obtained for the 100th iteration).

Gait appearance

Figure 5.3 presents several snapshots of one step obtained with the parameters. This gait is robust, i.e. the robot is able to walk without falling even with the noise on the stimulation signals.

There are some differences with a human gait, which make this gait look not so natural. First, due to the relatively large foot of the robot (compared to the one in Geyer's model), the foot is set flat on the ground and the heel and the ball touch the ground approximately at the same time. Conversely, when a human walks, his heel strikes the ground first and then the foot rolls on the heel until it is flat on the ground. This is the behaviour obtained with Geyer's model and also in the previous Biorob students' works [Ber11], [Dze13]. Besides, the trunk of the CoMan is tilted a little too much backward.

In figure 5.3, for $t = 5.56s$, we can see that the CoMan's right foot is very close to the ground. This is the main reason leading the CoMan to fall in the optimizations, i.e. when one leg in swing phase touches the ground before the end of the swing state. Indeed, the large foot is not lifted efficiently. Conversely, in Hartmut Geyer's model, the foot is smaller (proportionally to the body), so it is more difficult for the foot on this model to hit the ground.

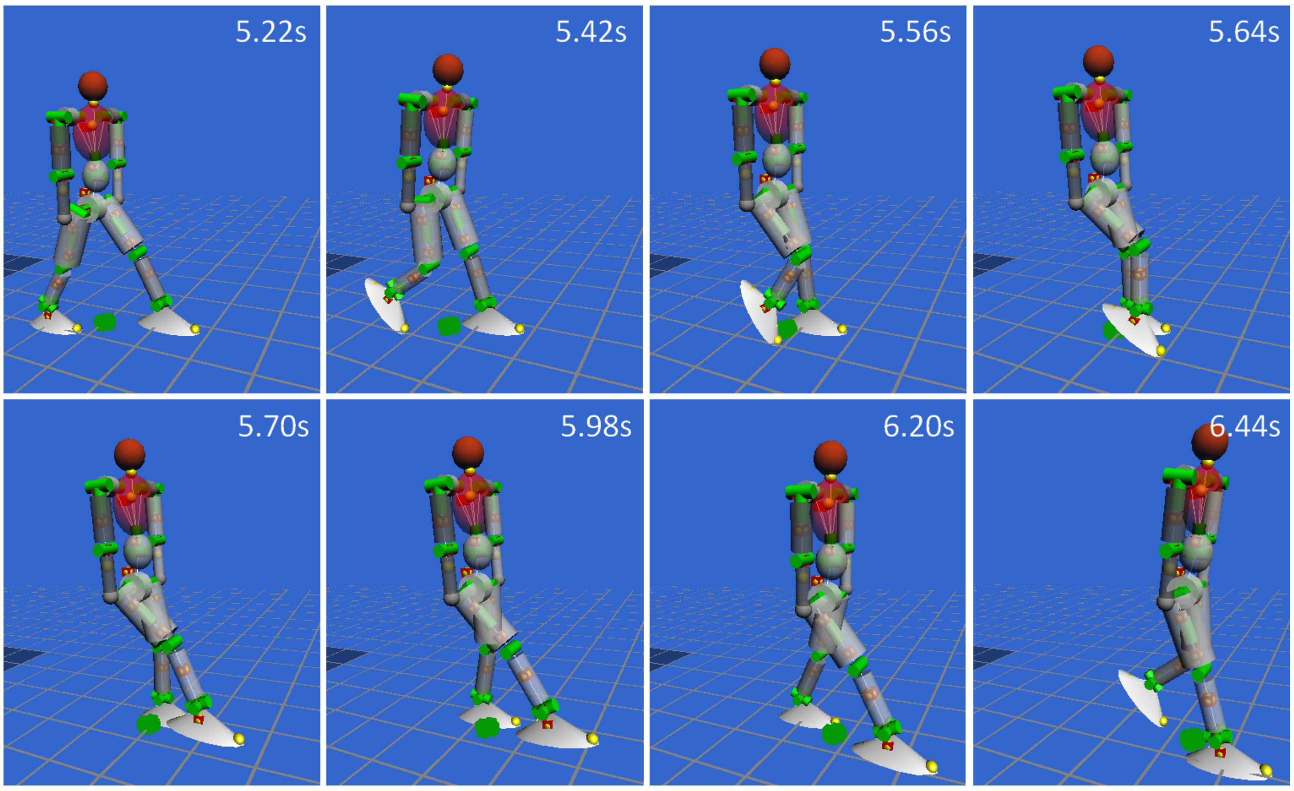

Figure 5.3: One step of the stable robust gait obtained after the optimization.

Figure 5.4 shows the waist height versus time. This describes a very regular pattern, except at the beginning where there is the gait initiation. This pattern repeats itself indefinitely since the gait is stable. This is the same for the torques generated by Geyer's model, i.e. the torques applied at the sagittal degrees of freedom of both legs. Figure 5.5 presents this torques for the right leg. This figures repeats itself indefinitely, too.

There is a huge peak in the hip and knee torques just before $t = 7s$, corresponding to the moment where the heel strikes the ground. At this moment, the controller switches from the "swing" state to the "stance" state, so there is a sudden change in the reflex rules used, which can partly explain the peak. On top of that, these torques allow the trunk to keep a correct inclination during the walk and, as stated before, the CoMan trunk is tilted a little too much backward with this gait. So, to prevent the trunk to fall backward, the torques needed are greater than what should normally be required with a more natural-looking gait.

On the real CoMan, the maximum torque produced by the actuators is 50 Nm. Here, we see in figure 5.5 that this limit is respected. However, in the tests, we limited the torques to a much lower value, to not risk to damage the actuators.

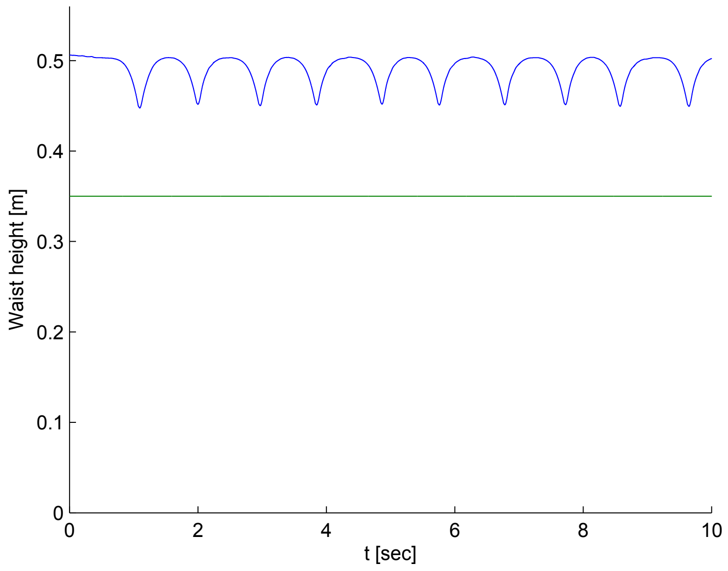

Figure 5.4: Waist height during the first 10 seconds of the walk (curved line in blue) and threshold to detect fall (flat line in green).

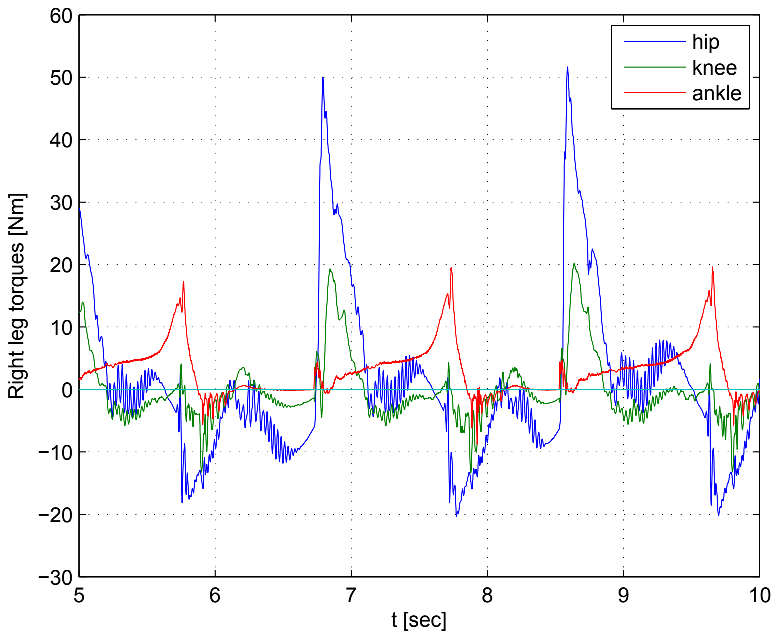

Figure 5.5: Torques of the sagittal joints of the right leg (same values for the left leg).

Chapter 6

From Simulation to Reality

The final goal of this master project is to implement the controller we develop in Robotran on a real robot: the CoMan. To achieve this, we use in simulation a model as close as possible to the real CoMan. Unfortunately, our controller is based on Harmut Geyer's work, who developed a controller for his own model: a 80 kg adult-like walker. The CoMan is more like a 4-year-old child and, consequently, we cannot use the parameters provided by Geyer in his paper [GH10]. So, the first part of this section describes how we can scale all these parameters with a method called dynamic scaling.

The next step is to look at the inputs and the outputs available on the real CoMan and then to use them to make a good interface between the controller we developed in simulation and the one on the real CoMan. We will see that with a good interface, it is possible to transfer the controller from simulation to the real CoMan without having to modify the files.

6.1 From Geyer's model to the CoMan: dynamic scaling

In Harmut Geyer's paper [GH10], lots of parameters are provided like the muscle maximal forces, the neutral length of the contractile elements (CE) and the series elastic (SE) elements, their maximal velocities, etc. These data were gathered to fit an adult-like model of 80 kg.

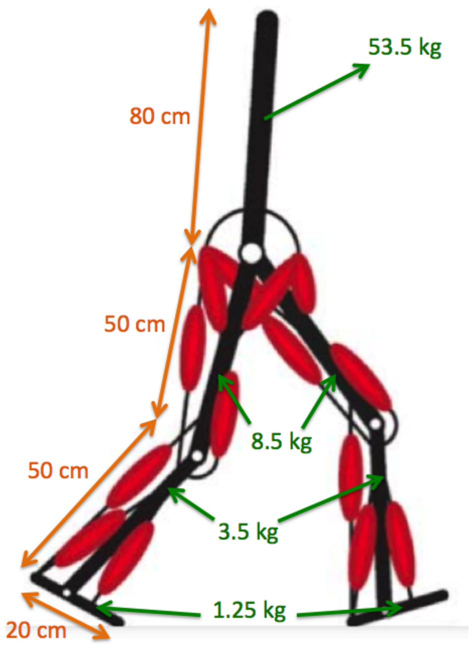

Figure 6.1: Harmut Geyer's model with segment lengths and masses

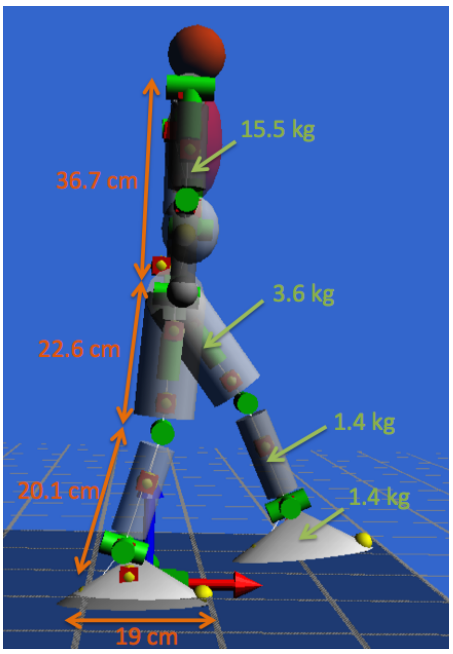

Figure 6.2: model of the CoMan with segment lengths and masses

Unfortunately, as we can see when we compare this model in figure 6.1 with the one of the CoMan in figure 6.2, the CoMan is more like a 4-year-old child and has different weights and lengths for the different parts of the body. All the parameters provided by Geyer do not fit this model. To ensure that the dynamic behavior of the CoMan matches the one of Geyer's model, we will use dynamic scaling to adapt these parameters.

The idea is that dynamic scaling uses fundamental physical variables (mass, length, and time) to define relationships between a system quantity at different scales. To do this, we will focus on the length and mass ratios. Even if these ratios depend on the different parts of the body, we will take a unique ratio for the lengths and for the masses. This is of course not strictly equivalent but provides a good approximation.

Moreover, this only means that the CoMan has a different morphology than the one from Geyer's model. For instance, we do not expect that all people on Earth would have the same muscles parameters if we scaled them. This is an exciting part of this work: because these muscles are virtual muscles, we can design them as we want and give to the CoMan stronger muscles if we want. All these choices will impact energy consumption, robustness,...

For the lengths ratio, we focus on the combined length of the thighs and the shanks ($20.1 + 22.6 = 42.7$ *cm* for the CoMan, $50 + 50 = 100$ *cm* for Geyer's model) while for the masses ratio, we focus on the total mass (28.3 *kg* for the CoMan, 80 *kg* for Geyer's model).

$$\frac{\text{CoMan lengths}}{\text{Geyer's model lengths}} \approx 0.427 \qquad \frac{\text{CoMan masses}}{\text{Geyer's model masses}} \approx 0.354$$

We must apply these parameters on neutral lengths, velocities, maximal forces and anchor points of the virtual muscles and on the torque reference used for the joint soft limit. This is immediate for the neutral muscle lengths (we only have to multiply the initial parameters by the lengths ratio 0.427). For the other parameters, we use basic physical laws. This method is described in [SGT12].

We can already notice an important point in the scaling: the feet sizes. While the other parts of the body have a lengths ratio of a bit less than 0.5 (between the CoMan and Geyer's model), the size of the feet in these two models are nearly the same (19 *cm* for the CoMan, 20 *cm* for Geyer's model). How can we explain this major difference?

None of these two models (CoMan and Geyer) have compliant feet. In fact, Geyer's model has little feet compared to the rest of the body while the feet of the CoMan are big compared to the rest of the body. So, none of these two models seem natural. The reason is the following: Geyer's model is devised for dynamic walking where a smaller foot can be easier (its shape is closer to a point and this model has the possibility to raise the leg less high because the foot in swing phase is less likely to hit the ground during this swing phase).

In fact, we have a robot with bigger non compliant feet. This is a bit similar to a human walking on a concrete road with non compliant shoes. For this reason, one of the different paths we will explore next semester is designing another foot which is more efficient for dynamic walking.

6.2 Inputs and outputs on the real CoMan

In simulation, there are two main parts: the controller and the physics which uses this controller to simulate the CoMan kinematics and dynamics. On the real CoMan, we only need to send torque or position references: these are the inputs on the real CoMan but correspond to the outputs of the controller we developed in simulation. Figure 6.3 shows these two parts and the variables that transit between them.

In figure 6.4 and 6.5 (which is a zoom on figure 6.4) we can see the effects of the position controller on the real CoMan, which is quite good. The continuous lines represent the actual joint positions while the discontinuous black lines represent their references. We have the same situation in figure 6.6 and in figure 6.7 (zoom on figure 6.6) but with torque control. We can see that the torque control on the sagittal degree of freedom of the left hip is also quite good. Nevertheless, we will see later that the torque control on the ankle is less efficient.

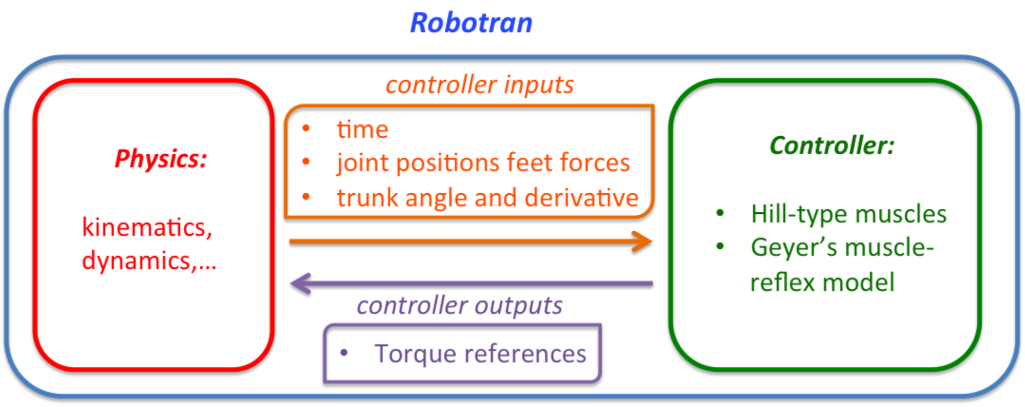

Figure 6.3: Inputs and outputs of the controller. Inputs of the controller are the outputs of the physics (the physical part of the CoMan model) and the torque references are the outputs of the controller, so inputs used by the physical model.

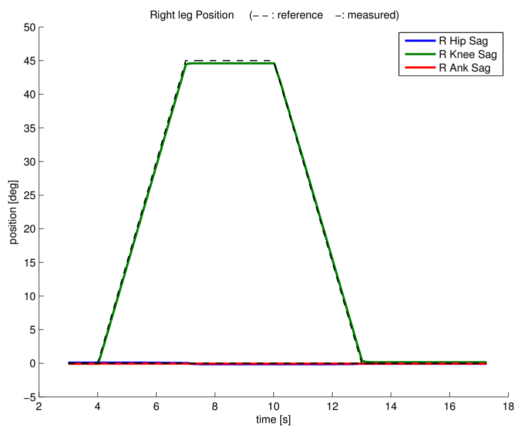

Figure 6.4: Position control applied on the right knee (real CoMan with IIT controller): position references (dash) and measured positions (continuous line).

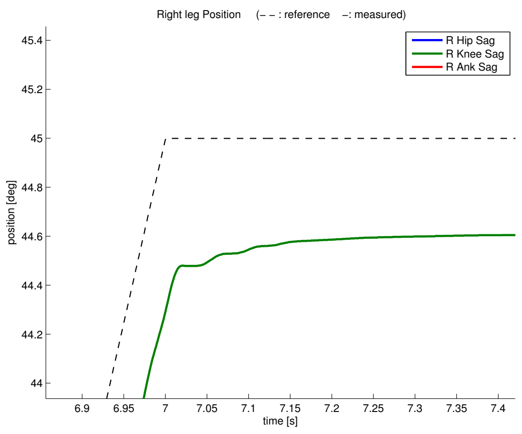

Figure 6.5: Zoom on figure 6.4.

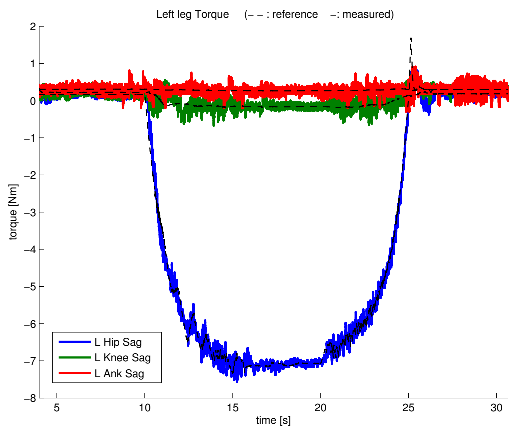

Figure 6.6: Torque control applied for an activation of the HFL muscle (real CoMan with IIT controller): torque references (dash) and measured torques (continuous line).

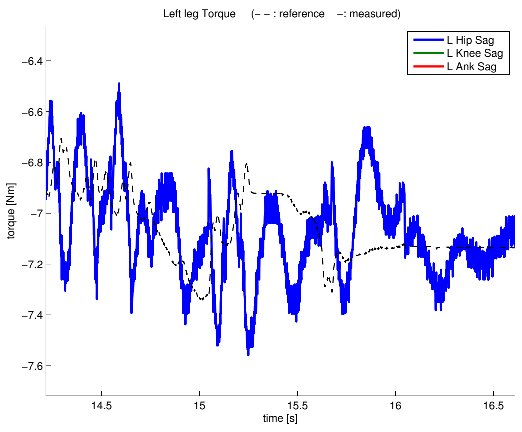

Figure 6.7: Zoom on figure 6.6.

As previously mentioned, there are two parts in the controller: the virtual Hill-type muscles and the activations computation with Harmut Geyer's muscle-reflex model, as we can see in figure 6.8.

For the Hill-type muscles, we need three inputs: a time reference, the position of the joints (which are outputs of the real CoMan) and the activations provided by Geyer's rules. The first two inputs are trivial, the only difficulty is to adapt the indexes and the frames which are not exactly the same in simulation and on the real CoMan. Thanks to these inputs, it is possible to get the torque references of all the sagittal joints of the legs.

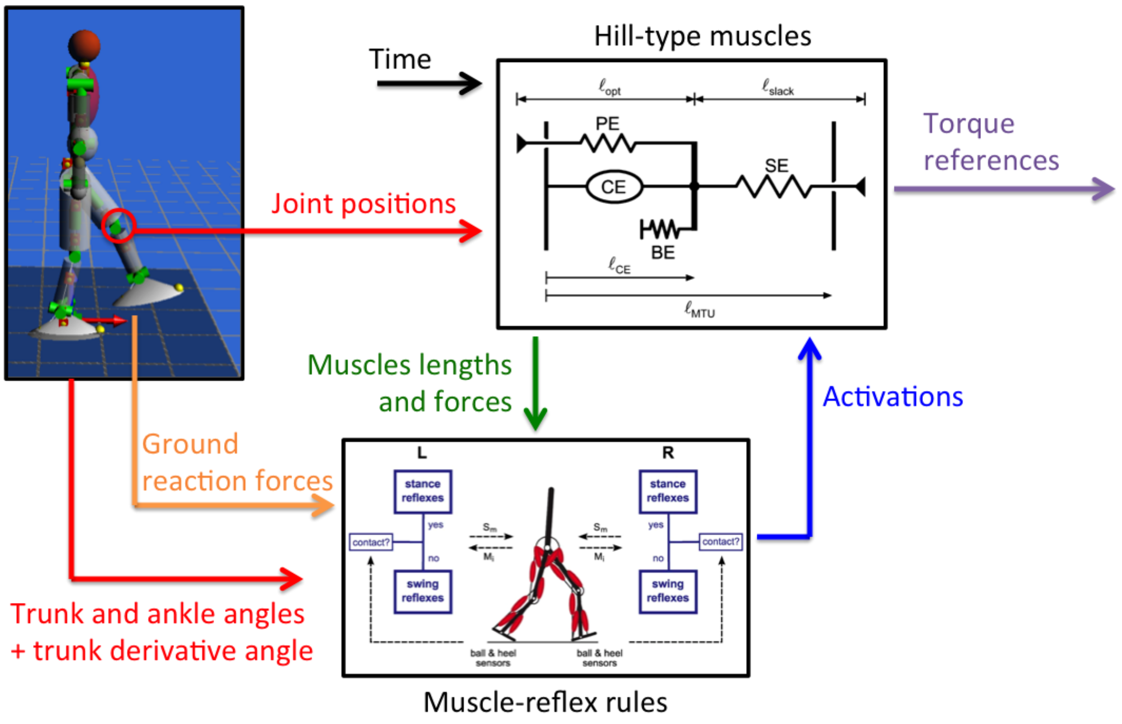

Figure 6.8: Relations between the different parts of the controller: the Hill-type muscles and the muscle-reflex rules. At the end, we get the torque references.

For Harmut's Geyer reflex muscles, two other inputs are necessary:

- *Ground reaction forces*
  The scalar value of the two ground reaction forces under the feet along the vertical axis are used by four virtual Hill-type muscles (HFL, GLU, HAM and VAS) during the *stance phase*. These forces are calibrated while the CoMan is hung in the air.

- *Angle of the trunk (relative to the gravitational axis) and its derivative*
  There is an Inertial Measurement Unit (IMU) in the trunk of the CoMan. Thanks to this, it is possible to derive the angle of the trunk and its derivative. Unfortunately, the morphology of the CoMan is different from the one in Geyer's model. In this model, the trunk is directly attached to the legs as we can see in figure 6.9 while with the CoMan there is a horizontal shift of approximately 2 *cm* between the legs and the trunk axes, as we can see in figure 6.10.

  Even if a 2 *cm* difference may seem small, this makes a very big difference during walking (especially during walk initiation) because Geyer's model is looking for the angle between the attachment point of the legs and the center of mass of the trunk. To solve this problem, we compute the angle of a virtual trunk which connects the attachment point of the legs with the center of mass of the trunk. This induces a constant shift of $3.77°$ in the angle measured by the IMU. For the derivative of this angle, no correction is necessary because the shift correction is a constant angle.

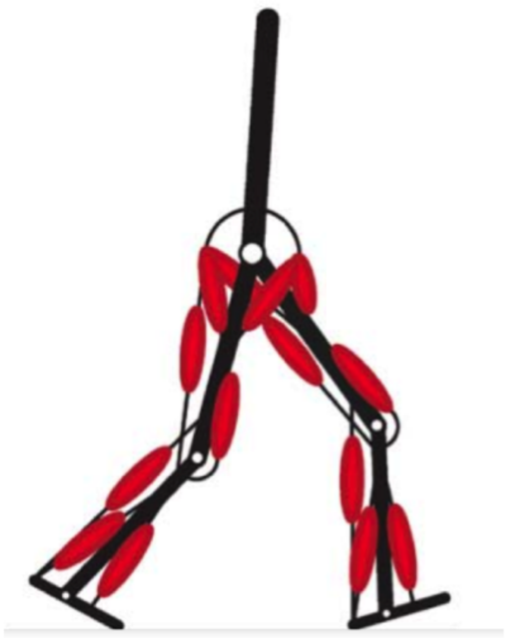

Figure 6.9: Harmut Geyer's model

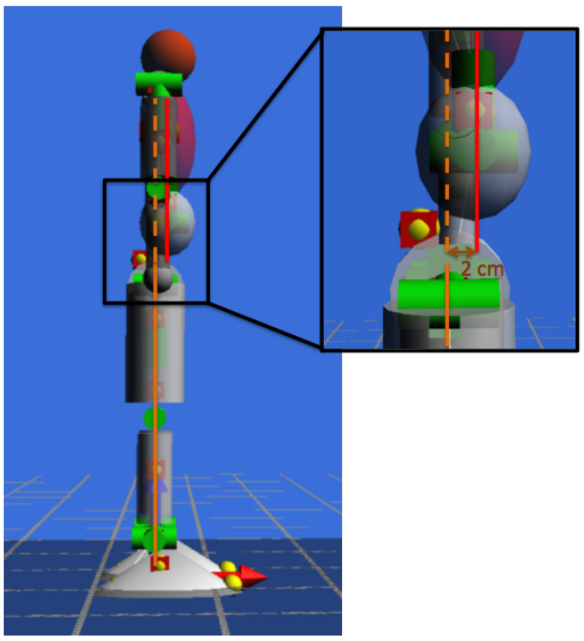

Figure 6.10: morphology of the CoMan

6.3 Controller: from Simulation to the real CoMan

The robot model has been written in such a way that this is easy to transfer the controller from simulation to the real CoMan. To this end, the two parts of the robot model (the physics with Robotran and the controller with Geyer's muscle-reflex model) have been clearly divided, as we can see in figure 6.11 and as developed in chapter 4.

The inputs and outputs of the controller have been described in section 6.2. The controller in simulation is implemented in the following way: an interface function (`user_compute_output`) receives the inputs from the CoMan (time, joint positions, ...). Then, it calls a function called `HG_main` which contains the Hill-Geyer's model. Finally, it sends the outputs (torque references) back to the controller. `HG_main` is a short function which calls the different functions of the controller.

To transfer the controller from simulation to the real CoMan, `user_compute_output` must be adapted to receive its inputs from the real CoMan (and not from Robotran) and to send the torque references computed by Geyer's model to the torque controller already implemented in the real CoMan. `user_compute_output` is a kind of interface and is the only function to modify. All the other functions (`HG_main` and its subfunctions) can be "copied-pasted" without any modification. Thanks to this, it is very easy to develop a controller in simulation and then to test it on the real CoMan. More details on this topic can be found in Appendix A.

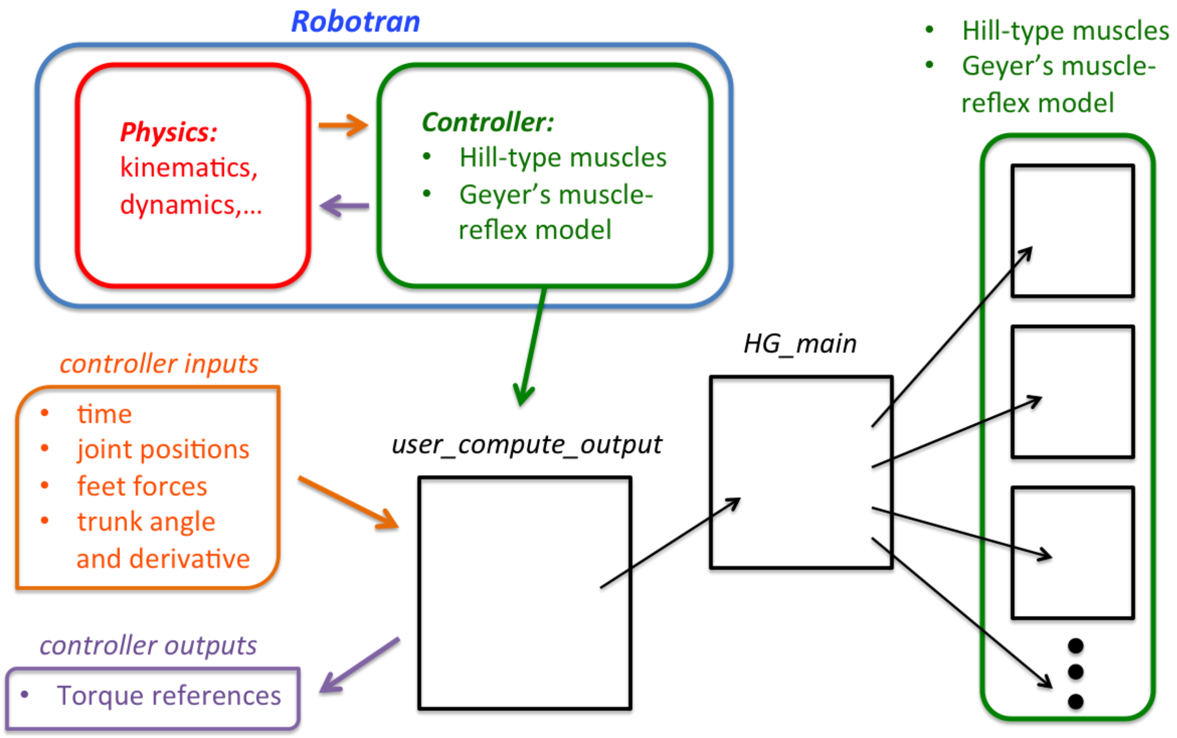

Figure 6.11: Implementation scheme of the simulation code

Chapter 7

Results & Discussion

After the transfer of the controller from simulation to reality (detailed in the previous chapter), it is now possible to look at the results and to compare the results we got from simulation to the ones from the real CoMan. As developed in section 3.1, there are two main parts in the implementation of this controller: the Hill-type muscles and the muscle-reflex model. We will start by testing the Hill-type muscles while the CoMan is hung in the air and then we will have a short look at the CoMan trying to initiate walking on the ground with Geyer's muscle-reflex model.

7.1 Hill-type muscles

We now have all the tools available to perform tests on the real CoMan. The aim of this section and of the next section is to study these results and to compare them with the predicted results we got from our simulation model in Robotran.

For the first tests, we want to test the virtual Hill-type muscles implemented on the real CoMan by sending arbitrary stimulations (not computed with Geyer's muscle-reflex model). To do this, the CoMan is hung in the air with two ropes attached at its neck, as we can see in figure 7.2. This configuration was reproduced in simulation (see figure 7.1). The two circles at the top of this figure represent the points were the ropes are attached on the structure. We impose a constant distance between the centers of these circles and two anchor points located at the neck of the simulated CoMan in order to reproduce the effect of the ropes. Thanks to this, we get a simulation environment very close to reality.

Three tests will be performed on the virtual Hill-type muscles. For the first test, we only stimulate one muscle, namely the hip-flexor muscle (HFL). Then, we will do the same for all muscles, one after the other. Finally, we will do the first experiment once again, but we will apply an external force to see how the CoMan reacts to it.

7.1.1 Activation of one muscle

As explained, we want to test the Hill-type muscles implemented on the real CoMan. These muscles (7 muscles for each leg) are controlled by the stimulations we send. We start by applying the minimal activation (0.01) on all muscles of both legs. This results in very small torque references (but not exactly zero-torque). Then, after 10 seconds, we linearly increase the stimulation of the HFL muscle in the left leg while all other stimulations remain fixed at 0.01. After 5 s, the stimulation of this left HFL muscle reaches its maximum value of 1.0 and we keep this constant value for 5 s. Then there is a linear decrease of this stimulation in order to reach again the minimal stimulation after 5 s. This trapezoidal shape of this stimulation is shown in figure 7.3. For the rest of this section, we will call the time when this HFL-muscle stimulation is above 0.01 the *activation period* (from $t = 10\,s$ to $t = 25\,s$) while the rest of the time will be called the *resting period.*

We will now study the positions and the torques of the 3 sagittal joints of one leg. The frames of the positions of these sagittal joints are indicated in figure 7.4. The colors are the same as the ones used in all graphs of this section (for both the position and the torque graphs). All these graphs show only the three sagittal joints for one leg.

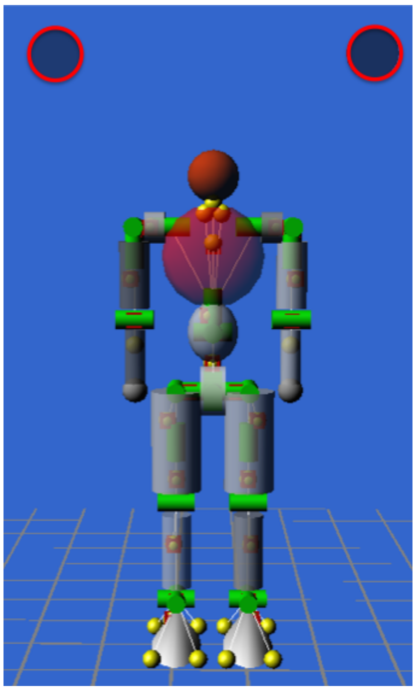
Figure 7.1: Simulated CoMan (Robotran) hung with two ropes (constrains on constant distances) at the neck

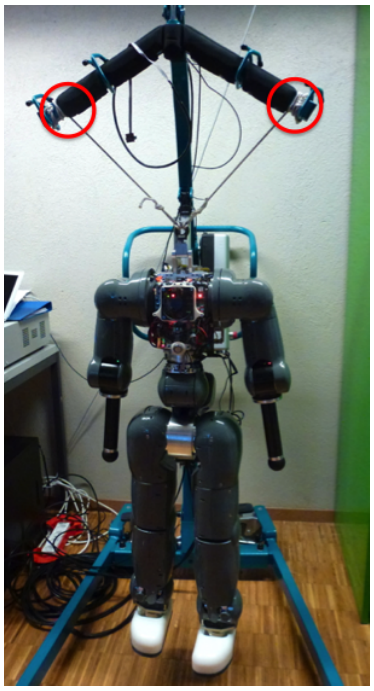
Figure 7.2: Real CoMan hung with two real ropes at the neck

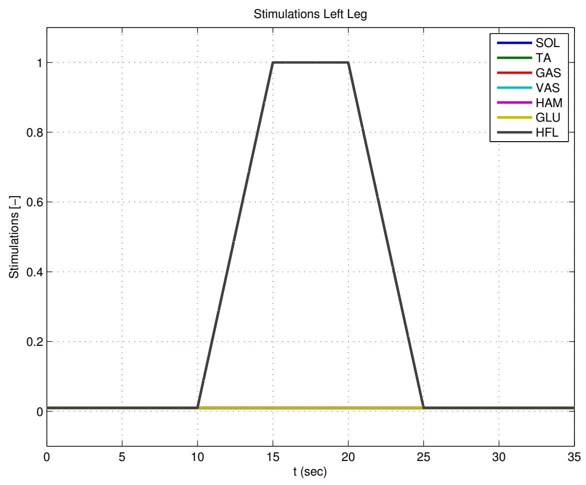

Figure 7.3: Stimulations sent to the 7 Hill-type muscles of the left leg (only HFL has a value different from 0.01 during the activation period)

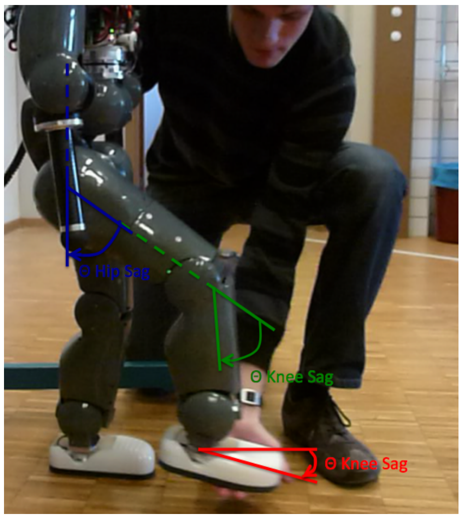

Figure 7.4: References angles of the 3 sagittal joints of each leg

Figure 7.5 represents the configuration of the simulated CoMan when the stimulation has reached its higher value. In figure 7.6, we can see the same for the real CoMan.

The evolution of the angles of the 3 sagittal joints of the left leg can be seen in simulation in figure 7.7 and on the real CoMan in figure 7.8. The results for the hip are quite similar (in simulation and on the real CoMan) during the activation period even if we do not apply a position control but a torque control on the joints derived from the muscles activations. Nonetheless, the matching between simulation and reality is not so good during the resting period.

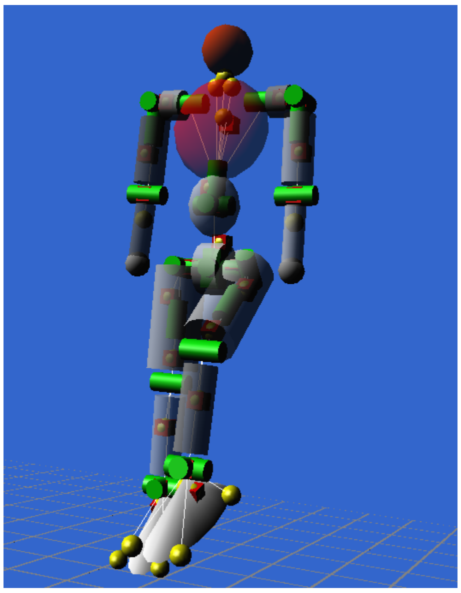
Figure 7.5: Configuration of the simulated CoMan (hung by virtual ropes) when the stimulation has reached its higher value (1.0).

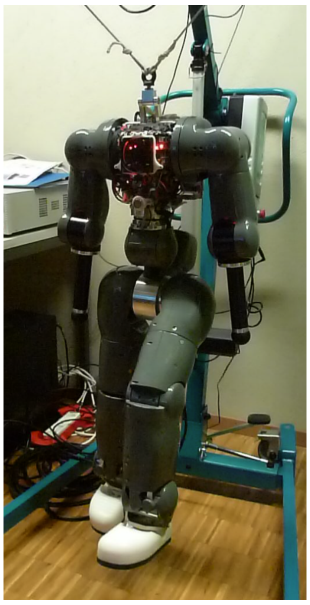
Figure 7.6: Configuration of the real CoMan (hung by real ropes) when the stimulation has reached its higher value (1.0).

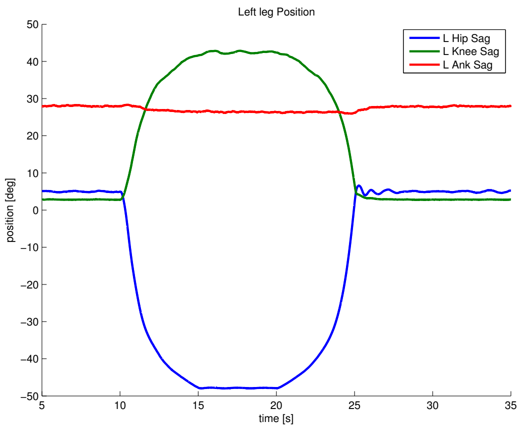

Figure 7.7: Position of the sagittal joints of the left leg in simulation (Robotran) when the HFL muscle is activated with a trapezoidal shape.

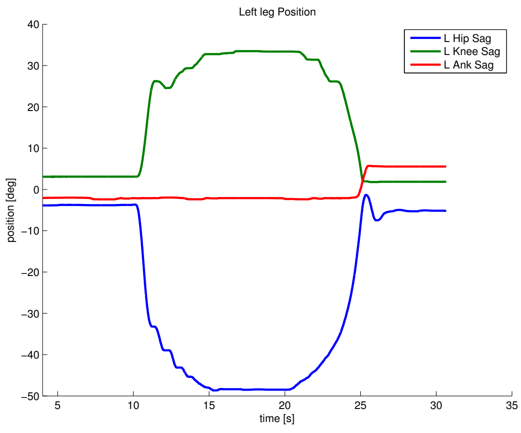

Figure 7.8: Position of the sagittal joints of the left leg on the real CoMan when the HFL muscle is activated with a trapezoidal shape.

One of the possible reasons to explain this is the friction between the different armature parts of the real CoMan body which are not modeled in simulation. Indeed, the torque reference is near 0, so the forces are low and the effects of the frictions have a bigger impact. This is particularly apparent when we look at the ankle which is hung due to its weight in simulation while in reality the low weight of the foot is compensated by the friction and remains approximately in its *zero-position* configuration. Furthermore, we see on the real CoMan that the ankle in torque control is very sensitive to the torque calibration and is far from being perfect. It might be interesting to modify the gains for the ankle. This is something we could see in simulation: gains needed to be higher on the ankle than on other joints to have a good torque control. Currently, all the torque gains are the same on the real CoMan. When we compare the figures 7.5 and 7.6, we can see that the position of the foot is indeed not the same for both models.

Finally, for the knee position during the activation period (see figures 7.7 and 7.8), the shape is approximately the same but still a bit different in magnitude (the peak is above 40° in simulation while it is under 40° on the real CoMan). Once again, this is related to the problems mentioned above: because the torque reference is close to 0, only the gravity is acting on this joint.

Figure 7.9 represents the torques and their references in simulation (black discontinuous line for the references) for the sagittal joints of the left leg while we have the same graph in figure 7.10 for the real CoMan.

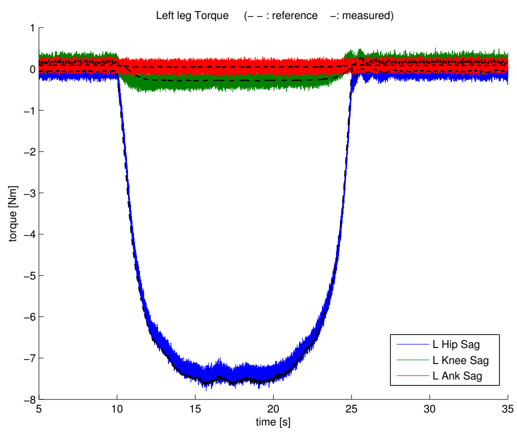

Figure 7.9: Torque references (discontinuous black lines) and real torques of the sagittal joints of the left leg in simulation (Robotran), same experiment as the one described in figure 7.7

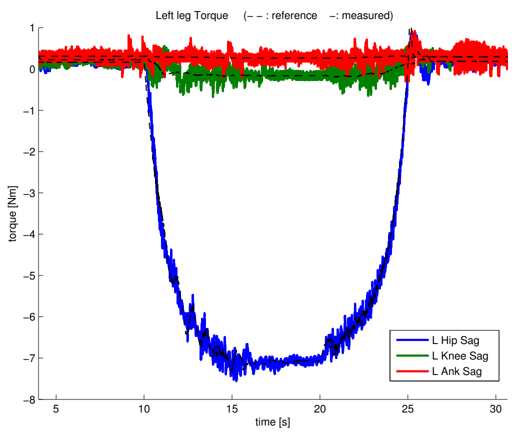

Figure 7.10: Torque references (discontinuous black lines) and real torques of the sagittal joints of the left leg on the real CoMan, same experiment as the one described in figure 7.8

Basically, the results are quite similar for both graphs (simulation and reality), except that the real CoMan seems more shaky. Some non idealities non implemented in simulation (friction,. . .) can explain these differences but there are two main other reasons that can explain this. First, the controller is not exactly implemented in the same ways in both cases, we will still work on this during the next semester. The second reason is that the torques are not measured in the same way: there is a torque sensor on each joint of the real CoMan while we measure the elongation of the series elastic elements in the motors to get the torques in simulation. A possible improvement for the next semester is to use a similar measurement for the simulation model and the real CoMan.

7.1.2 Activation of the 7 muscles

Now, we do the same test, but with all the 7 Hill-type muscles. Once again, we use the same trapezoidal shape for the different stimulations as the one we saw in figure 7.3. Here again, one muscle is activated while the others keep a minimal activation. We start by activating one muscle during 15 s (5 s for linear increase, 5 s for constant higher value and 5 s for linear decrease), then there are 5 s where all the stimulations are 0.01. Then, we repeat the same scheme, but with another muscle. The order of the muscles for the stimulations (and their associated sagittal joints) is the following: HFL (hip); GLU (hip); HAM (hip and knee); VAS (knee); GAS (knee and ankle); TA (ankle); SOL (ankle), all for the left leg.

The results can be seen in figure 7.11 in simulation and in figure 7.12 for the real CoMan. Once again, there is a good matching between simulation and reality for the joints with a torque reference which is not close to 0. For instance, the blue lines represent the hip torques and show a good matching at the beginning when the muscles acting on this joint are activated. Then, after 60 s, the matching is not so good anymore because the torque reference of the hip is close to 0. We have something similar with the ankle (in green) when the corresponding muscles are activated.

We can see that the torque reference for the sagittal degree of freedom of the ankle (last muscles activated) is a bit shaky on the real CoMan and less in simulation. The torque control seems better in simulation than on the real CoMan. Here again, we can see that using the same gains for the torque control on the ankle as the ones used on the other joints is not optimal.

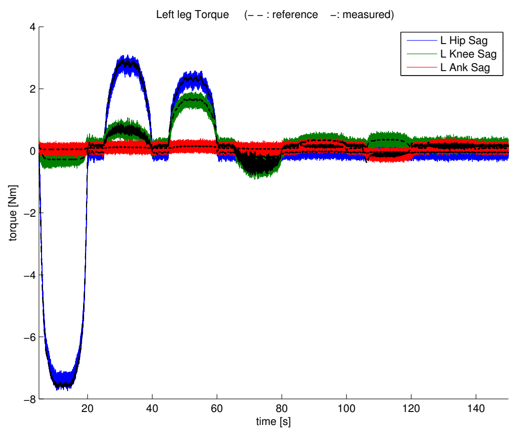

Figure 7.11: Torque references (black lines) and real torques of the sagittal joints of the left leg in simulation when we activate all muscles, one after the other

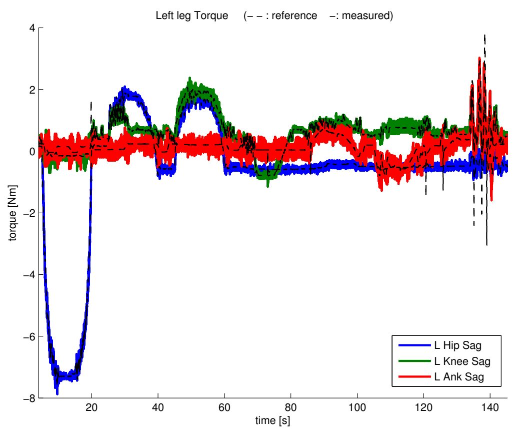

Figure 7.12: Torque references (black lines) and real torques of the sagittal joints of the left leg on the real CoMan when we activate all muscles, one after the other

7.1.3 Perturbations on one activated muscle

In this experiment, we only activate one muscle: the hip-flexor muscle (HFL), as for the first experiment, but this time, we keep the maximal activation during 25 s instead of 5 s and we interact during this time with this leg to see how the CoMan reacts. The kinds of interactions done on the CoMan can be seen in figures 7.13 and 7.14.

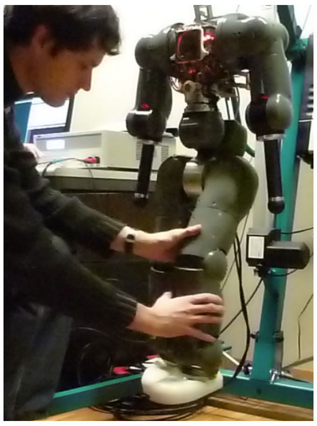

Figure 7.13: Interaction on the left leg while the HFL virtual muscle has a maximal stimulation

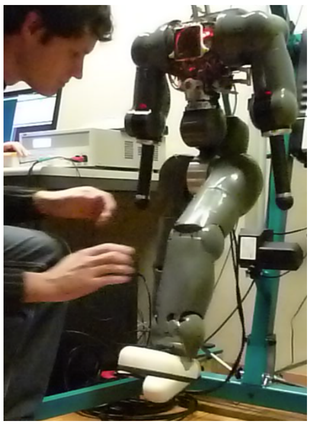

Figure 7.14: Same test as the one described in figure 7.13, but a short while after (less than 1 s)

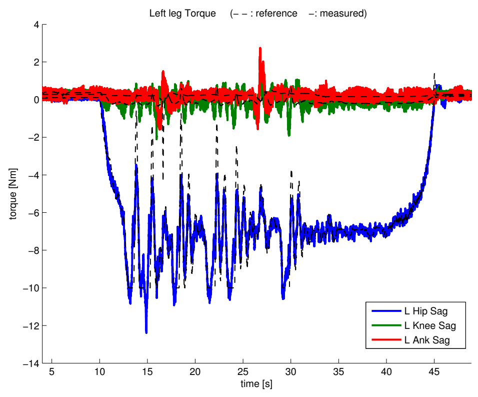

Figure 7.15: Torque references (black discontinuous lines) and real torques of the sagittal joints of the left leg on the real CoMan while there is an external interaction with its body (as in figures 7.13 and 7.14)

In figure 7.13, the CoMan is resisting while an external force (human hand pression) is applied to counteract the work done by the HFL left virtual muscle. In figure 7.14, the external force disappears and this HFL muscle is released and can raise the hip joint and then can stabilize it as before the interaction. In figure 7.15, we see that the hip torque reference (the black discontinuous line) adapts its behavior to the external force to ensure that we keep a behavior similar to the one we would have on a real human with a maximal stimulation on the HFL muscle.

The peaks of this reference torque correspond to new external forces acting on this leg. Typically, we have a higher absolute value for the torque reference at the beginning of a new external force to counteract this external force and then a lower value when this force is released. After 32 s, there is no more interaction and the hip torque reference stabilizes near the same value as in figure 7.10 (the same experiment but without external forces).

7.2 Hartmut Geyer's muscle-reflex model

The aim of the previous section was to test the virtual Hill-type muscles implemented on the real CoMan. Now, we want to test the second part of the controller: the computation of the stimulation signals with Hartmut Geyer's muscle reflex model. Two tests will be studied in this report.

The first test is related to walk initiation while the second aims to test the reflexes on one foot with no contact with the ground while the other is still in contact with the ground, just after the walk initiation.

7.2.1 Walk initiation on the ground

To perform this test, the feet force sensors are calibrated while the CoMan is hung in the air. Then, we start the Hill muscles model and we put the CoMan on the ground. The CoMan, which is upright, starts by touching the ground with a small slack in the ropes which are attached to its neck. At time $t = 25\,s$, we start Hartmut Geyer's muscle-reflex model. Because we already started the Hill-muscles model before, the lengths of the contractile elements (l_{CE}) already converged, as explained in section 3.1.1.

When Geyer's model is activated, we can see that one leg starts the swing phase while the other remains on the ground for the stance phase. Nevertheless, there is only a small slack in the ropes which are attached to the neck of the CoMan. We decided to do this for security reasons and because the model is still not able to walk in 3D. As a consequence, these ropes prevent the foot in swing phase to touch the ground, the chain of reflexes is interrupted and the CoMan keeps the same position.

This experiment is still sufficient to get some interesting results on walk initiation. As was mentioned in section 3.2, the first leg in swing phase is not always the same. In both simulation and on the real CoMan, we can see this phenomena. The figures 7.16 and 7.17 show the CoMan in walk initiation with the left leg which is the first in swing phase while in figures 7.18 and 7.19, it is the right leg which is in the first in swing phase.

The reason of this behavior was explained in section 3.2: it depends on the feet forces sensors. The leg with the biggest force will stay on the ground and initiate the *stance phase* while the other will start the *swing phase*.

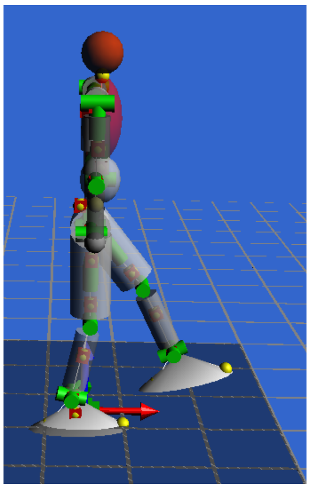
Figure 7.16: Left leg first in swing phase - simulation (Robotran)

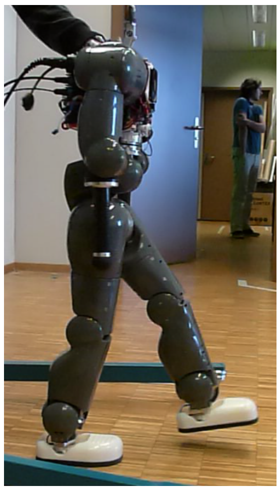
Figure 7.17: Left leg first in swing phase - test on the real CoMan

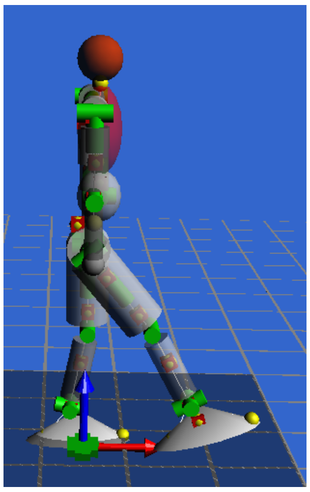
Figure 7.18: Right leg first in swing phase - simulation (Robotran)

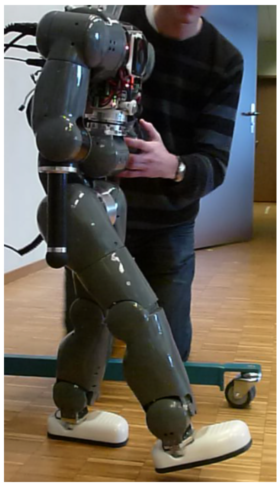
Figure 7.19: Right leg first in swing phase - test on the real CoMan

We can see this in figure 7.20, 7.21 and 7.22 which represent the situation described in figure 7.17 (when the CoMan initiates its walk with the left leg in swing phase). Figure 7.20 shows the feet forces while figures 7.21 and 7.22 show the torques (and their references in discontinuous lines) of the sagittal joints of the left leg (figure 7.21) and of the right leg (figure 7.22).

In figure 7.20, we can see that at $t = 25\,s$ (when Geyer's rules are activated), the right foot force is higher than the left one. As a consequence, the left leg will start the swing phase while the right one will stay on the ground for the stance phase. We can see the torque references for this swing phase of the left leg in figure 7.21. There is at beginning a high peak on the hip torques to flex this hip in order to initiate walking. Figure 7.22 is a bit more complex to analyze because reflexes during the stance phase are more complex (see in [GH10]) and involve complex inputs like the feet forces.

Then, we compare these results with the ones from figures 7.23, 7.24 and 7.25 which represent the same graphs but for the situation described in figure 7.19 (right leg is the first in swing phase). Here, we have the opposite: at $t = 25\,s$, the left foot force is higher than the right (see figure 7.23) and the right leg starts the swing phase. We have similar patterns in figure 7.21 on the left torques (when the left leg is in swing phase) and in figure 7.25 on the right torques (when the right leg is in swing phase). Indeed, these patterns depict the beginning of the swing phase. Even if these two situations are not identical, these figures show that the patterns at the beginning of the swing phase are quite similar, no matter which leg initiates walking.

We also have a similar pattern when we compare figures 7.22 and 7.24, but is less similar because it represents the beginning of the stance phase where we have more complex equations and more complex inputs like feet forces.

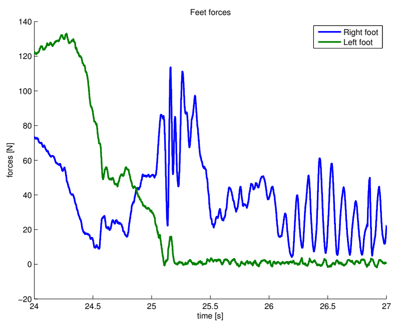

Figure 7.20: Feet forces when the left leg initiates swing phase (see figure 7.17)

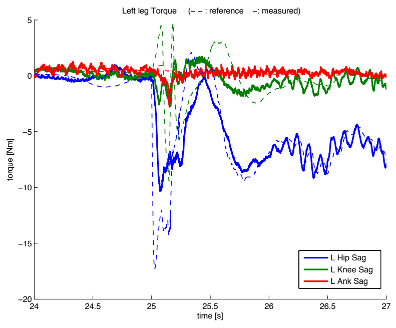

Figure 7.21: Left leg torques and their references (discontinuous lines) when the left leg initiates swing phase (see figure 7.17)

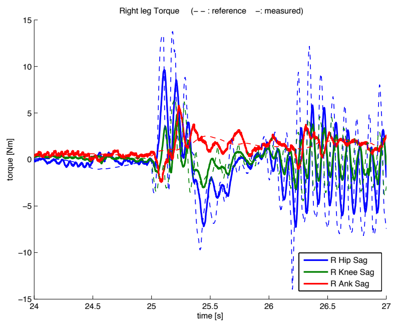

Figure 7.22: Right leg torques and their references (discontinuous lines) when the left leg initiates swing phase (see figure 7.17)

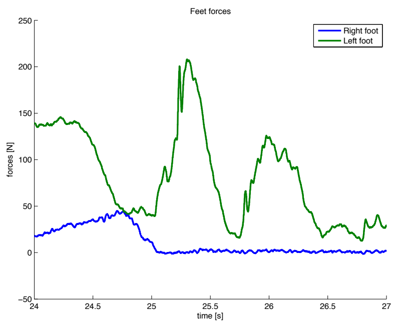

Figure 7.23: Feet forces when the right leg initiates swing phase (see figure 7.19)

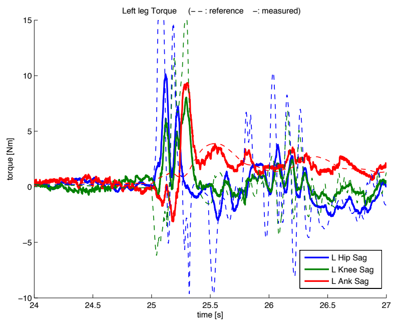

Figure 7.24: Left leg torques and their references (discontinuous lines) when the right leg initiates swing phase (see figure 7.19)

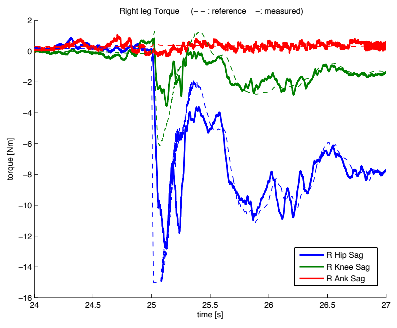

Figure 7.25: Right leg torques and their references (discontinuous lines) when the right leg initiates swing phase (see figure 7.19)

7.2.2 Testing the reflexes on one foot with no contact with the ground

This experiment starts by the same way as the one described in 7.2.1 but this time, we test the reflexes of the leg in swing phase. As we can see in figure 7.26, the walking initiation has already been done and the left leg is on the ground while the right leg is straight and waits to touch the ground to go on with the reflex chain. At this moment, this chain is blocked because of the ropes which prevent the right foot from touching the ground.

Figures 7.26 to 7.31 show what happens when we apply a brief force under the right foot to simulate the contact with the ground. We can see that the hip and the knee react in a similar way as what they would do in case of real ground contact.

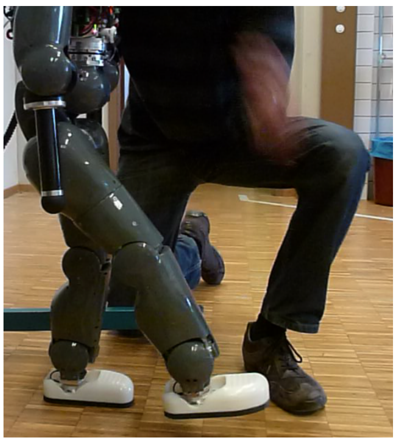
Figure 7.26: Reflexes test on the right leg - current time: $t = 29.0\,s$

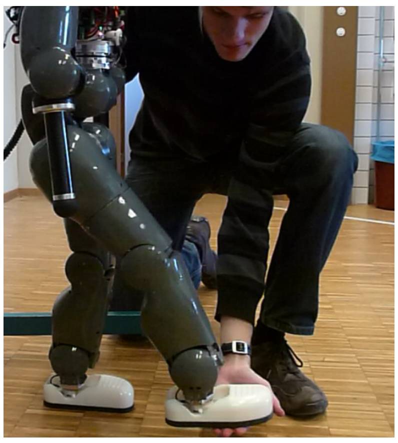
Figure 7.27: Reflexes test on the right leg - current time: $t = 29.5\,s$

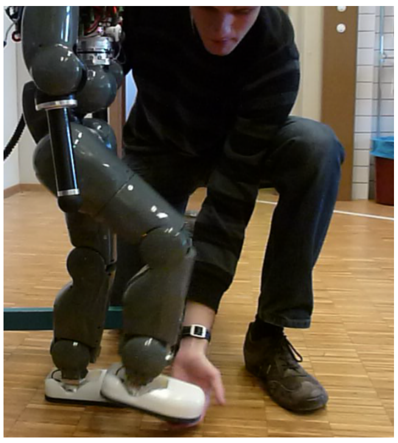
Figure 7.28: Reflexes test on the right leg - current time: $t = 30.3\,s$

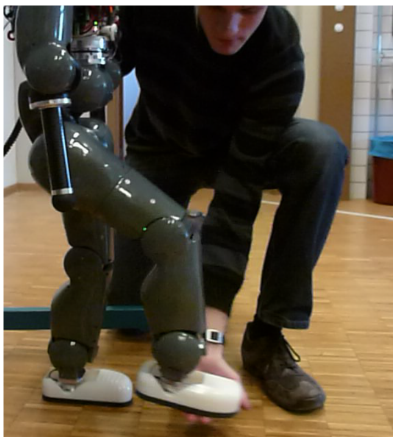
Figure 7.29: Reflexes test on the right leg - current time: $t = 30.5\,s$

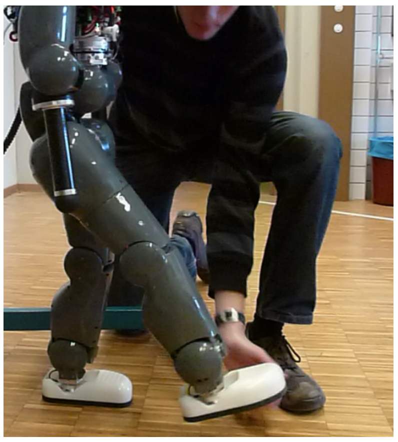
Figure 7.30: Reflexes test on the right leg - current time: $t = 30.7\,s$

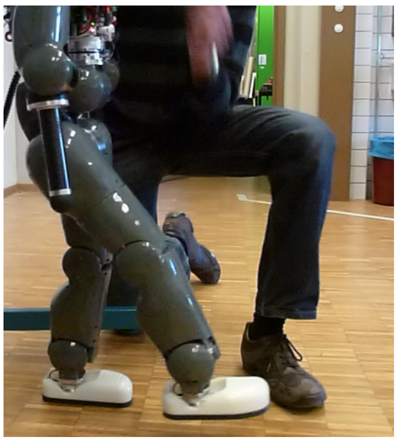
Figure 7.31: Reflexes test on the right leg - current time: $t = 31.4\,s$

Figure 7.32 shows the positions of the sagittal joints of the right leg while figure 7.33 shows the corresponding torques and their references (discontinuous lines). We can see that these reflexes start at $t = 29.5\,s$ (the first moment when the external force is applied). The knee angle was previously small (see figure 7.32), but at this precise moment, the CoMan tries to flex this knee by applying a higher torque reference on the knee joint, as we can see in figure 7.33 with the green line (this reference was previously negative to maintain the leg straight and counteracts the gravity effect, now this reference becomes positives to flex the knee).

Concerning the hip, we previously had to apply a huge torque (in absolute value) to raise it when the leg was not in contact with the ground (see the blue line in figure 7.33). When there is a contact with the ground (in this case a human hand), this torque reference decreases (in absolute value) because the CoMan knows that the hip angle must adapts itself to the ground (the effect of the ground will be to raise the hip, so there is no need anymore to apply a big torque to do this).

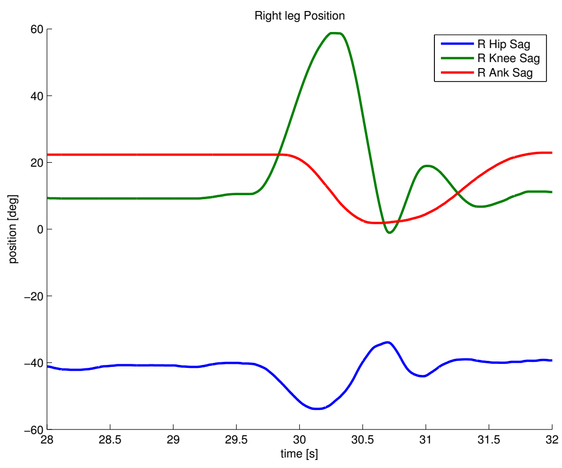

Figure 7.32: Joint positions of the sagittal joints of the right leg during the external interaction described in figures 7.26 → 7.31.

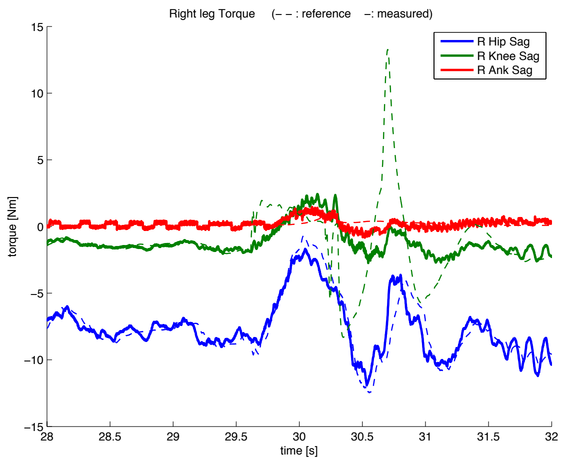

Figure 7.33: Joint torques and their references (discontinuous lines) of the sagittal joints of the right leg during the external interaction described in figures 7.26 → 7.31.

Chapter 8

Conclusion & Perspectives

8.1 Conclusion

We managed to get a robust, 2-dimensional walking controller in simulation. We were able to transfer this controller on the real robot by reusing the same code thanks to a well-defined interface between the controller and the physics in the CoMan model. Then we performed a first series of validation tests on the real robot.

The results of these tests can be separated into two classes. On the one hand, we have differences between the simulated CoMan and the real one. On the other hand, we have differences between the walking gait obtained with the walking controller developed for the CoMan and the results given by Geyer's model.

We can explain the differences between the real and simulated CoMan by the following reasons. First, the model of the actuators is not exact and there is an additional friction between the different segments of the CoMan which is not taken into account in the model. Next, the torque control in the real CoMan is very sensitive to the torque calibration and this can introduce some errors between simulation and reality.

Considering the walking gait we got after the optimization phase for the CoMan, we noticed a qualitative difference in its appearance compared to the one obtained with Geyer's model. The gait obtained with Geyer's model looks more natural than our, with the heel striking the ground before the ball. In our case, the robot puts its foot flat on the ground.

The causes of that behaviour are twofold. First, we use a real robot, with a real body, which means swinging arms, e.g. In the same idea, the CoMan body presents some differences with the robot model used in Geyer's model: on the one hand, the trunk is not aligned with the legs but horizontally shifted by a few centimeters and on the other hand the foot of the CoMan is relatively larger than the one used in Geyer's model, which has a great influence on the gait appearance.

The other key difference between Geyer's model and the real CoMan is that the torque references cannot be applied exactly in reality. Indeed, the torques are produced by actuators, which must be controlled. There are thus always unavoidable non-idealities in the torque controllers that introduce errors in the system.

In addition to developing the controller for the dynamic walking, we also had to develop the robot model starting from a model received from IIT. We translated this model into C-code which enabled us to run fast simulations and then optimizations of the controller. The robot model simulates the dynamic behaviour of the robot and the results are close to reality thanks to a precise modelling of the different elements of the robot (inertia of the segments, actuators models, etc.).

This was made possible by using the Robotran simulation environment, which is based on a multibody approach. This is the underlying framework on which we based the development of our model.

8.2 Perspectives

We have now another semester to implement a complete, three-dimensional walking controller for the CoMan. There are lots of open paths in front of us and we try here to give an overview of the avenues to explore.

The ideas developed below will first be addressed in simulation, as we come back in Belgium for the next semester. Nevertheless, we intend to take advantage of the Easter break to come back at Biorob to run the tests on the real CoMan and try to make it walk !

First, we can improve the model of the actuators in the CoMan model by taking into account the current consumed. We can also implement the new version of the actuator controller model given by IIT in the late December.

Moreover, we can continue the model development by freeing ourselves from the Matlab/Simulink environment. The idea is to have a lighter model to be able to run the optimizations faster.

Thereafter, the next big step will be the development of the 3-dimensional walking controller. Indeed, the CoMan is currently still constrained to move in the sagittal plane, the other DOF being frozen. This will necessitate a new literature review because this is an entirely new topic on its own.

Furthermore, we also have the possibility to derive a new foot for the CoMan. The idea is to test and compare the walking performances of the CoMan with different foot designs. Indeed, we noticed that the foot shape has a big influence on the robot gait.

Another topic could be to focus on the initiation and the end of the walking gait. Here, we managed to make the robot start walking from home position but this is not optimal yet and moreover, we never took care about the ending of the gait.

Finally, an interesting topic is to use CPG in association with reflexes to modulate the walking speed of the CoMan and recover the same gait after a small perturbation.

Appendix A

Controller structure

The figure A.1 shows the organization of the code of the controller used with the real CoMan. There is only one function per file, with the same name as the file itself. As stated in the report, most of the controller files can be reused without modification. Nevertheless, there are some changes to do for the following files:

- `user_compute_output.c` becomes `CoMan_compute.c`
- `coman_def.h`
- `MBSdataStruct.h`
- `user_sf_IO.h`
- `user_sf_IO.c`

The modifications consist in adapting the files and functions to the robot, i.e. removing all the code related to the simulator itself. Indeed, with the real robot, the parameters measured by the sensors come directly from the robot but in the simulator these parameters are measured by virtual sensors, which are functions. This kind of code must be removed. For `user_sf_io (.h` or `.c`), we have to remove the I/O related to the Simulink model and keep only the internal variables (also defined in these files).

The different functions are called in `HG_main` in the order suggested in figure A.1 (from top to bottom). Concerning the function `HG_main_lower_step`, it is the implementation of the concept explained in section 3.1.1, in the paragraph "Hill-type muscles: integration time step". This explains that the controller must be integrated with a smaller time step than the physical model of the CoMan to let the muscle lengths converge properly.

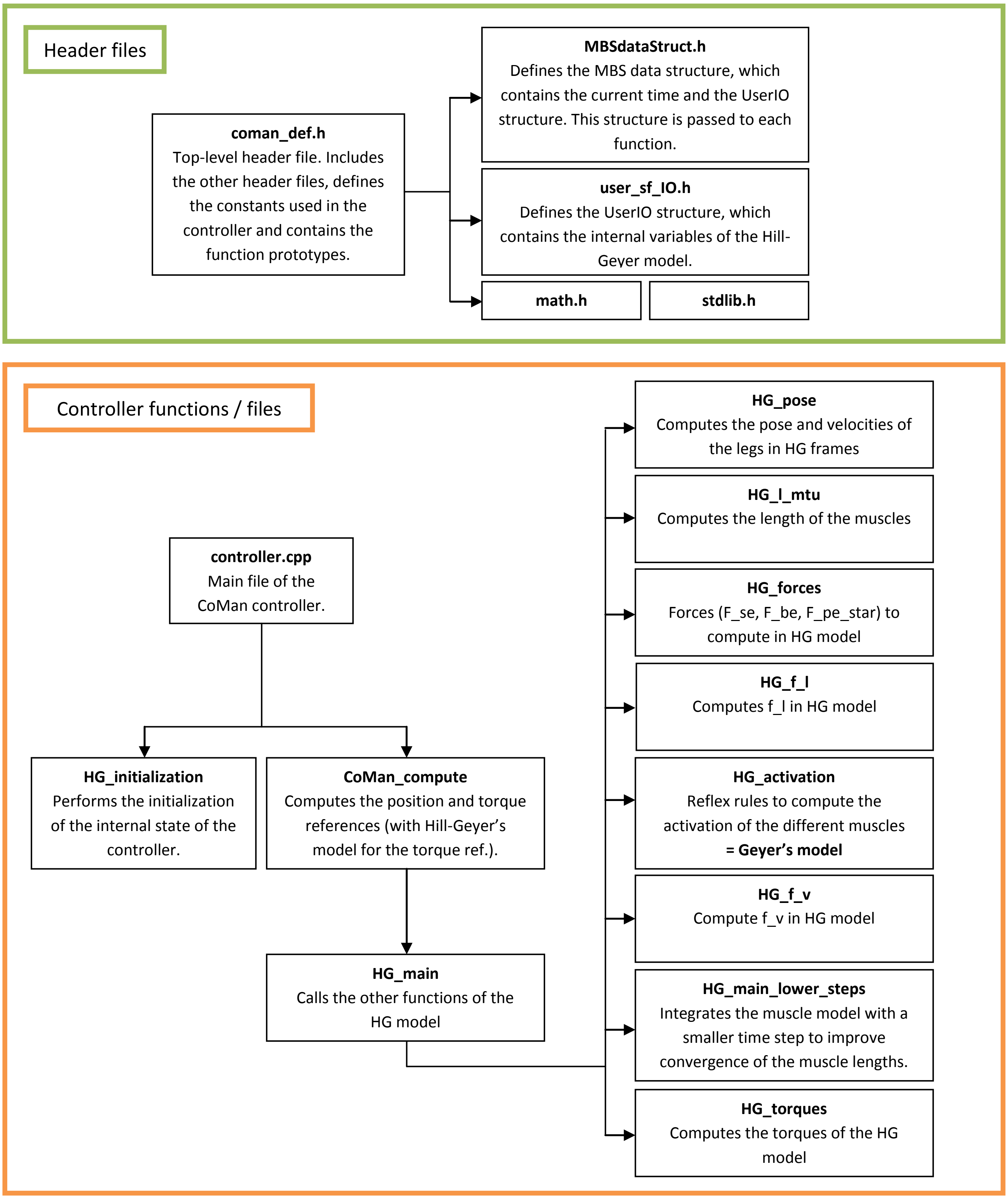
Header files
coman_def.h
Top-level header file. Includes the other header files, defines the constants used in the controller and contains the function prototypes.
MBSdataStruct.h
Defines the MBS data structure, which contains the current time and the UserIO structure. This structure is passed to each function.
user_sf_IO.h
Defines the UserIO structure, which contains the internal variables of the Hill-Geyer model.
math.h
stdlib.h
Controller functions / files
controller.cpp
Main file of the CoMan controller.
HG_initialization
Performs the initialization of the internal state of the controller.
CoMan_compute
Computes the position and torque references (with Hill-Geyer's model for the torque ref.).
HG_main
Calls the other functions of the HG model
HG_pose
Computes the pose and velocities of the legs in HG frames
HG_l_mtu
Computes the length of the muscles
HG_forces
Forces (F_se, F_be, F_pe_star) to compute in HG model
HG_f_l
Computes f_l in HG model
HG_activation
Reflex rules to compute the activation of the different muscles
= Geyer's model
HG_f_v
Compute f_v in HG model
HG_main_lower_steps
Integrates the muscle model with a smaller time step to improve convergence of the muscle lengths.
HG_torques
Computes the torques of the HG model

Figure A.1: Controller structure.

Appendix B

Bibliography

[Bar10] A. Barrea. *Robotran sous Simulink*, October 2010.

[Ber11] S. Berger. Energy consumption optimization and stumbling corrective response for bipedal walking gait. Master's thesis, Ecole Polytechnique Fédérale de Lausanne (EPFL), Biorob Laboratory, Lausanne, June 2011.

[CER09] CEREM. *Modeling Multibody Systems with ROBOTRAN*. `http://www.robotran.be/`, September 2009.

[CER12] CEREM. Robotran homepage. `http://www.robotran.be/`, 2012.

[CK02] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58 – 73, February 2002.

[DMMC+12] H. Dallali, M. Mosadeghzad, G.A. Medrano-Cerda, N. Docquier, P. Kormushev, N.G. Tsagarakis, Z. Li, and D.G. Caldwell. Development of a dynamic simulator for a compliant humanoid robot based on a symbolic multibody approach. IIT, 2012.

[Dze13] F. Dzeladini. (unknown). Master's thesis, Ecole Polytechnique Fédérale de Lausanne (EPFL), Biorob Laboratory, Lausanne, January 2013.

[FSW08] A. Faisal, L.P. Selen, and D.M. Wolpert. Noise in the nervous system. *Nature Reviews Neuroscience*, 9:292 – 303, April 2008.

[GH10] H. Geyer and H. Herr. A muscle-reflex model that encodes principles of legged mechanics produces human walking dynamics and muscle activities. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 18(3):263 – 273, 2010.

[GSB03] H. Geyer, A. Seyfarth, and R. Blickhan. Positive force feedback in bouncing gaits? *Proc. R. Soc. Lond. B*, 270(1529):2173–2183, October 2003.

[GSB04] H. Geyer, A. Seyfarth, and R. Blickhan. Spring-mass running: simple approximate solution and application to gait stability. *J. Theor. Biol.*, pages 315 – 328, 2004.

[IIT12] IIT. Coman hardware. COMAN Meeting IIT, January 2012.

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942 – 1948, November/December 1995.

[LT07] D. Liu and E. Todorov. Evidence for the flexible sensorimotor strategies predicted by optimal feedback control. *The Journal of Neuroscience*, 27(35):9354 – 9368, August 2007.

[McM84] T.A. McMahon. *Muscles, Reflexes, and Locomotion*, chapter Chap. 9 - Effects of Scale, pages 234 – 247. Princeton University Press, 1984.

[MMCS$^+$12] M. Mosadeghzad, G.A. Medrano-Cerda, J.A. Saglia, N.G. Tsagarakis, and D.G. Caldwell. Comparison of various active impedance control approaches, modeling, implementation, passivity, stability and trade-offs. In *Advanced Intelligent Mechatronics (AIM), 2012 IEEE/ASME International Conference on*, pages 342 – 348, July 2012.

[Ron12] R. Ronsse. Humanoids, 2012. Cours MECA2732 Robotics, UCL.

[SF03] J.-C. Samin and P. Fisette. *Symbolic modeling of multibody systems.* Dordrecht ; Boston : Kluwer Academic Publishers, 2003.

[SGT12] A. Schepelmann, H. Geyer, and M. Taylor. Development of a testbed for robotic neuromuscular controllers. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[SK08] Siciliano and Khatib, editors. *Springer Handbook of Robotics*, chapter 56. Springer, 2008.

[TLSC11] N.G. Tsagarakis, Z. Li, J. Saglia, and D.G. Caldwell. The design of the lower body of the compliant humanoid robot "cCub". In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2035 – 2040, May 2011.

[TMDMC12] N.G. Tsagarakis, S. Morfey, H. Dallali, and G.A. Medrano-Cerda. COmpliant HuMANoid Platform (COMAN). `http://www.iit.it/en/advr-labs/humanoids-a-human-centred-mechatronics/advr-humanoids-projects/compliant-humanoid-platform-coman.html`, 2012.

[Tod05] E. Todorov. Stochastic optimal control and estimation methods adapted to the noise characteristics of the sensorimotor system. *Neural Computation*, 17(5):1084 – 1108, 2005.

[UCL12] CEREM UCL. Robotran tutorial. Presentation of the Robotran framework, 2012.

[vdK11] J. van den Kieboom. *Optimization Framework Conceptual Overview.* Biorob EPFL, `http://biorob.epfl.ch/page-36418-en.html`, September 2011.