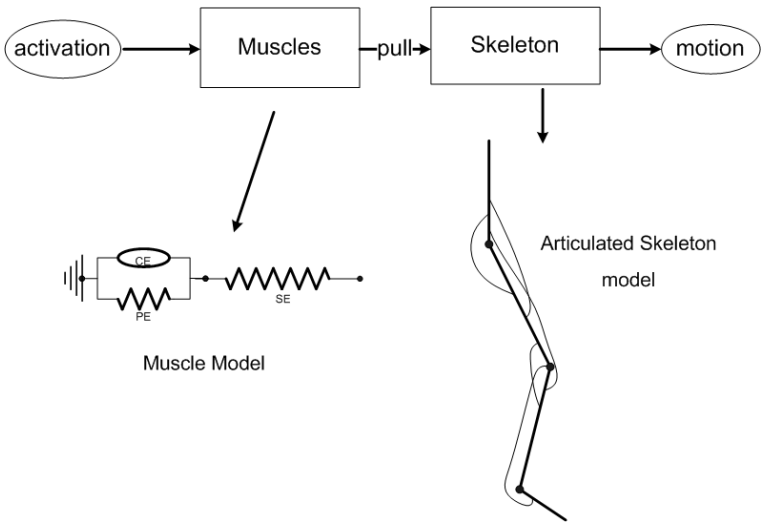
Modelling Martial Arts Techniques

J.C.K. Wong



A Thesis submitted for the degree of Master Scientific Computing
Supervisor: Dr. Gerard L.G. Sleijpen

Department of Mathematics, University Utrecht

October 2007

Contents

Chapter 1

Introduction

We want to let the computer find optimal movement as from martial art techniques. We should model human motions and optimize these motions. It involves mathematical modelling, programming and optimization. In this chapter we look how we can formulate and solve these problems in the most general sense.

1.1 Martial arts

Since the beginning of mankind, people have to protect themselves against others or wild animals. Sometimes with weapons and sometimes barehanded. Through centuries people have discovered that some movements give better results than other. Martial arts techniques can be seen as special movements, meant to defend or attack. Martial arts are the arts of adapting these techniques to ones self and using it creatively. Martials arts nowadays are mostly practiced for better health, which is also a way to protect ourselves.

1.2 Building a musculoskeletal model

We are interested in the motion of a human body. The human body is a very complex biological structure and consists of many systems. The system we are interested in is the locomotor system. The locomotor system consists of a skeletal system and a muscular system that drives the skeletal system. Both are very dynamic systems. The locomotor system is therefore known as musculoskeletal system. To bring down complexity we will only model part of the total musculoskeletal system, the leg with its skeletal muscles for example. In chapter 2 we will see how we can build a skeletal model in Simulink with SimMechanics. In chapter 3 we will use Virtual Muscle to build skeletal muscles and add them to the skeletal model. The number of skeletal muscles depends on the complexity of the skeletal model. After this we can start simulation and eventually extend the model. The input is an activation signal for each skeletal muscle. These signals stand for the neural activation of the muscles. Each simulation produces a certain motion. A martial arts technique is a optimal motion for a given goal. We thus have to optimize the model.

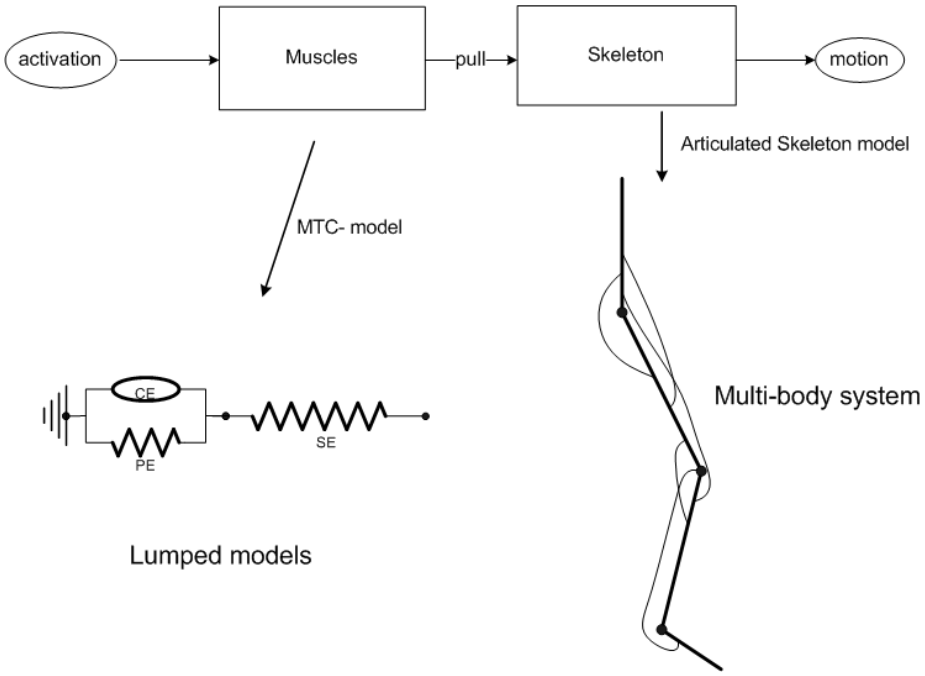


Figure 1.1: Schematic overview of how we try to find martial arts techniques. We model the human locomotor system, that has neural activation as input and will produce motion. A martial arts technique is an optimal motion with respect to a certain goal. If we want to find an optimal motion we should find the neural activation that produces this optimal motion. The locomotor system consists of 2 very different subsystems: the muscular and skeletal system. The muscular system is made up with a number of skeletal-muscles, which are actually muscle-tendon complexes(mtcs). They are activated by the neural activation. As result of the neural activation the skeletal-muscles will contract and produce a pulling force. This will cause the skeletal system to change and we get motion. We will model the skeletal muscles from a mechanical point of view and these type of models are known as lumped models. The skeletal system will be modelled with a system of rigid bodies that can only rotate with respect to each other(articulated models). Because the skeletal model is a system of multiple bodies, it can be considered as a multi-body system.

1.3 Modelling dynamic systems

With modelling we mean mathematical modelling. A mathematical model is a simplification of a real world system. It uses a set of equations to describe the system. The complexity of the equations depends on the system and how real we want it. For dynamic systems we have 3 main types of time-dependent variables:

Input variables $\vec{U}(t)$: User given input. The variables are known.

State variables $\vec{X}(t)$: The minimum number of variables that describe the state of the system at a certain time.

Output variables $\vec{Y}(t)$: These are the variables we are interested in. They depend on the input and state variables. We have $\vec{Y}(t) = g(\vec{U}(t), \vec{X}(t))$

The state of a dynamic system changes over time and the rate of change depends on the input. In physical systems the current state typically depends on "previous" states. The type of equations that describe the relations are the well-known ordinary differential equations:

$$\frac{d\vec{X}}{dt} = \dot{\vec{X}}(t) = f(\vec{X}(t), \vec{U}(t)) \tag{1.1}$$

1.3.1 Simulation

If we want to study the behavior of a dynamic system over a certain time T, we should simulate the model. This is done by specifying the state at the beginning $\vec{X}(0)$ and solving equation 1.1. $\vec{X}(0)$ is also known as *initial conditions* vector.

$$\begin{cases} \vec{X}(0) = \vec{X}_0 \\ \int_{t=0}^{T} \dot{\vec{X}}(t)dt = \vec{X}(t) \end{cases} \tag{1.2}$$

1.3.2 Numerical solvers

Equation 1.2 can only be solved in some very simple cases. In general we should use numerical methods to solve it. The easiest way to understand how this can be done is with the *Euler method*. With Taylor series we know that:

$$\vec{X}(t_n + \triangle t) = \vec{X}_{n+1} = \vec{X}_n + \dot{\vec{X}}(\triangle t) + O(\triangle t^2) \tag{1.3}$$

Euler method uses the first order Taylor series to approximate $\vec{X}(t_{n+1})$:

$$\vec{X}_{n+1} \approx \vec{X}_n + \dot{\vec{X}}\triangle t \tag{1.4}$$

The error is thus of order $O(\triangle t^2)$ and this method is not very accurate. A more accurate method is the *midpoint method* or $Euler - Heun$ and goes as follow

$$\vec{k}_1 = f(\vec{X}_n, \vec{U}_n, t_n) \tag{1.5}$$

$$\vec{k}_2 = f(\vec{X}_n + \frac{1}{2}\vec{k}_1\triangle t, \vec{U}_{n+\frac{1}{2}}, t_n + \frac{1}{2}\triangle t) \tag{1.6}$$

$$\vec{X}_{n+1} = \vec{X}_n + \vec{k}_2\triangle t + O(\triangle t^3) \tag{1.7}$$

With this we have an error term of $O(\triangle t^3)$ and should be more accurate. Euler's method and the midpoint method belong to the family of *Runge-Kutta* methods. They are also known as the *first order Runge-Kutta* and *second order Runge-Kutta methods* respectively. Runge-Kutta methods use different approximations for $\vec{X}_n$ and combine them in such a way that a higher order error term is obtained. Higher order Runge-Kutta methods use more function evaluations than lower order but should be more accurate. A common way to bring down the computation costs is to use variable time-steps. Numerical integrators can be categorized into two main methods, *implicit* and *explicit*. Explicit methods use known state variables from the current state to calculate state variables for the next state, while implicit methods do not. We will use Euler's method to show the difference. The explicit Euler method is known as Euler forward method and calculates the next state as follows:

$$\vec{X}_{n+1} = \vec{X}_n + \dot{\vec{X}} \triangle t \tag{1.8}$$

The implicit form of Euler method is known as Euler Backward method and goes as follows:

$$\vec{X}_{n+1} = \vec{X}_n + \dot{\vec{X}}_{n+1} \triangle t \tag{1.9}$$

Implicit methods that are used in practice have better stability properties than explicit ones per time step, but they usually are more expensive [1].

1.4 Modelling and simulation with Simulink

In order to get numerical results we should implement the mathematical model on the computer and let the computer do the numerical integration. There are several ways to do this. Simulink is a very popular and user friendly platform to model and simulate multi-domain dynamic systems. For more information we refer to *www.mathworks.com/products/simulink*. A Simulink model is a block-diagram that represents the set of equations that describe the system. It also has a wide variety of built-in numerical integrators to do the simulation. Throughout this thesis we will use Simulink as the modelling environment for our musculoskeletal models and the ease of simulation with Simulink will become very clear.

1.4.1 Modelling with Simulink

In Simulink we use a block diagram to represent the equations of one model. The block diagram is made up of blocks and signals. A block can be an operator, constant, output etc. Signals send time-dependent values between the blocks. The values can be of different data-type (double, integer) and multi-dimensional. Relevant blocks can be grouped into a larger block that represents a subsystem for example. It is very common to build subsystems in a way that they have an one-on-one correspondence with real systems. By doing this we can get clear and easy to understand models [2].

1.4.2 Simulation with Simulink

To simulate the Simulink model we should specify an integrator and its accuracy. Simulink has several ready to use built-in numerical integrators. The

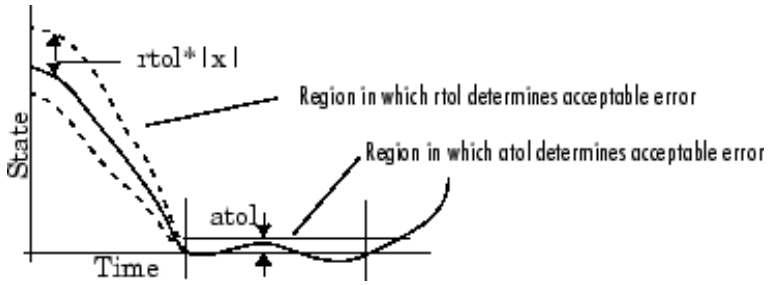


Figure 1.2: The specifications of regions where the approximated state variable will be accepted. Source: [2].

2 main types of solvers are fixed and variable time-step. In dynamic systems variable time-step methods are recommended. For real-time simulations, Simulink restricts us to use only fixed step solvers. When no real-time results are needed, the recommended choice is the ODE45 variable time-step solver with $relTol = 10^{-6}$. The ODE45 solver is based on the Runge-Kutta method of order 4 and is an explicit method. It is a variable time-step solver and its accuracy is set by specifying the relative tolerance $relTol$ and absolute $absTol$. The $relTol$ specifies in which region x_i is accepted and $absTol$ specifies the threshold when x_i is near zero. State x_i is only accepted if all errors e_i satisfy

$$e_i \leq \max(x_i * relTol, absTol) \tag{1.10}$$

otherwise the solver will use a smaller time-step. In figure 1.2 we see the different regions. If we set $absTol$ to automatic, Simulink will start with an $absTol = 10^{-6}$ and readjust it during simulation with $relTol$. At time n we have $absTol_i = relTol * \max(x_i(t))$.

1.5 Optimization problems.

We build a musculoskeletal model with different types of questions in mind. Most of these questions can be translated into optimization problems. These questions contain keywords like best, most, less, worst etc. For example: the best kick, the most economical kick. Optimization is a different field in mathematics. One type of optimization is to find best or worst output for any kind of input. We rate the output with an objective function $\mathcal{F}(Y)$. This function $\mathcal{F}$ tells us how good the results are. We name the function that maps the input to $\mathcal{F}$ h: $h(\vec{U}) = \mathcal{F}$(see figure 1.3). Our problem is to find an optimal input $(\vec{U})_{opt}$ such that $h(\vec{U}_{opt}) \leq h(\vec{U})$ or $h(\vec{U}_{opt}) \geq h(\vec{U}) \quad \forall \quad \vec{U}$ if we want to minimize or maximize respectively. It is very likely that $h(\vec{U})$ has many local optima. [1] We will implement an evolutionary algorithm that should be able to solve this type of optimization problems.

[1] We can experience this in martial arts practice too. We always find a better way to execute a technique.

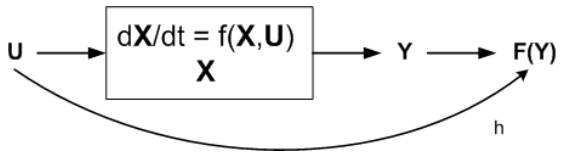


Figure 1.3: Schematic view of a dynamic system. The output $\vec{Y}$ depends on the state $\vec{X}$ and the input $\vec{U}$ and is rated with objective function $\mathcal{F}$. To get the state of the system we should use numerical integrators. Function h is very complex and with each simulation we get to know one more point of h. Our optimization problem is optimizing h.

Chapter 2

Building skeletal models

2.1 Introduction

In this chapter we will build skeletal models. First we take a look how we model the skeletal system as a mechanical system. Then we will study the dynamics of such a system. To get familiar with some basic mechanics we derive the equations of motions for a simple thigh-model. After that we will use SimMechanics to build skeletal models.

2.2 The skeletal model

The skeletal model is a sub-system of our musculoskeletal system. It will receive input from the muscle-models and produce motion. Moving objects on earth obey certain laws. These laws are known as Newton's laws of motion and have already been formulated in 1687 by Isaac Newton [3]. They are:

1. Every object continues in its state of rest, or of uniform motion in a straight line, unless it is compelled to change that state by forces impressed upon it.
2. The change of motion is proportional to the motive force impressed and is made in the direction of the line in which that force is impressed.
3. To every action there is always imposed an equal reaction; or, the mutual actions of two bodies upon each other are always equal and directed to contrary parts.

Newton's laws are only valid in an inertial frame. An inertial frame is a coordinate frame that does not undergo acceleration nor rotation. Real inertial frames do not exist but for many applications we can use a frame that is attached to the ground as an inertial frame.

2.3 Simplifications

In modelling it is always a tradeoff between complexity and realism. The more realistic we want the model, the more complex it will be. Below are the simplifications for our skeletal models:

2D motions: Only planar motions are considered. A main reason for this is the absence of data needed for 3D simulation.

Rigid bodies: We divide the human body into several segments and treat them as rigid bodies. The segments are connected with each other via rotational joints. Skeletal models are therefore known as *articulated* models. Rigid bodies are idealized physical objects and their mechanics are well understood.

Part of the body: we will only model and simulate parts of the body that undergo motions. A limb for example. There reason is that we need to build muscle models as well. The fewer bodies, the fewer muscle models are needed.

Fixed base and loose end: Our skeletal model can now be seen as a chain of rigid bodies. It has a base point and an end point. The base point is closer to the whole body than the end point. We fix the base and let the end loose. By doing this our skeletal models have only rotational degrees of freedom.

Simplified joints: the modelled joints allow pure rotational motions within a certain range. With real-life joints we have gliding and sliding [5].

With the above simplifications we can build a leg or arm model. A martial arts technique which can be simulated with a 2D leg model is the front-kick. With this technique we can kick a target in front of ourselves.

2.4 Dynamics of a rigid body in 2D

Before we start to build a skeletal model we will look at the dynamics of a rigid body. Dynamics studies the relation between motions and forces. A rigid body is an idealized physical object. It can be seen as a system of particles, where their relative distances are fixed. Real rigid bodies do not exist, but one might think of a brick or iron bar. The skeletal bones can be considered rigid as well[1]. If we want to study the planar motion of a rigid body we need to know the following properties of the rigid body first:

- mass m
- inertia moment I_O with respect to a certain point O.
- length l of the body
- center of mass CM by specifying p or d where
  - p length between the CM and the proximal end.
  - d length between the CM and the distal end. We have $p + d = l$

[1]The rigid bodies that models body segments includes the bone(s) and other organic materials around it such as muscles and skin.

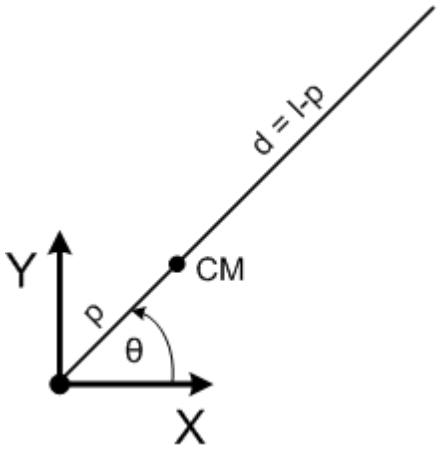


Figure 2.1: The thigh in an inertial frame. Because the thigh is heavier on the proximal side, p is smaller than d. The configuration of the thigh is determined by θ.

In figure 2.1 we have a sketch of a rigid body in an inertial frame. The CM is a very special point. The gravitational force $F_z = mg$ is modelled to act on this point[3]. The position of CM is denoted with $\vec{r}_{CM} = (x_{CM}, y_{CM})$ where:

$$\begin{cases} x_{CM} = \int \frac{xdm}{dm} \\ y_{CM} = \int \frac{ydm}{dm} \end{cases}$$

The CM depends on the shape of the body. If we have a full symmetric body and the body is made of uniform material, then its CM is located in the geometric center; for example this happens for an iron ball. The inertia moment I is usually determined with respect to the CM of the rigid body and is given by:

$$I_{CM} = \int r^2 dm \tag{2.1}$$

Here r is the perpendicular distance between an element of mass and point CM. I_{CM} tells us about the amount of resistance against rotational motion around CM. In many applications the rotation is around another point. If we name the point O and the distance between O and CM equals r we can calculate I_O with Steiner's parallel axis theorem:

$$I_O = I_{CM} + mr^2 \tag{2.2}$$

Note that Euler's equation(see next section) is only valid if O is fixed. This restriction does not hold for CM. Because of the quadratic contribution of r the shape of the body plays a very important role. In practice we can experience this. For example it is much easier to raise a chambered leg than a fully extended leg.

2.4.1 Equations of motion

An unconstrained rigid body in 2D has 3 degrees of freedom: 2 translational and 1 rotational. General motions of the rigid body can be seen as combinations of translational and rotational motions. The dynamics of the translational motions can be written with Newton's equation. Newton's equation of motion along an axis is:

$$\sum F = m\frac{d^2x}{dt^2} \tag{2.3}$$

With $\sum F$ the sum of all forces that acts along that axis and x is de coordinate of that axis.
Euler's equation can be used to describe the rotational motion with respect to CM:

$$\sum M = I_{CM}\frac{d^2\theta}{dt^2} \tag{2.4}$$

Here $\sum M$ is the contribution of all force-moments and torques with respect to CM. The force-moment is given with:

$$\vec{M} = \vec{r} \times \vec{F} \tag{2.5}$$

where $\vec{r}$ is the distance vector between CM and $\vec{F}$'s point of exertion. The perpendicular component of $\vec{r}$ on $\vec{F}$'s line of action is also known as the moment-arm of force $\vec{F}$. The larger the moment-arm the larger the effect of $\vec{F}$.
We speak of a torque $\vec{T}$ if two equally but opposite forces $\vec{F}_1$ and $\vec{F}_2$ act on the body with distance. If $\vec{d}$ is the distance vector between both forces we have:

$$\vec{T} = \vec{d} \times \vec{F}_1 = -\vec{d} \times \vec{F}_2 \tag{2.6}$$

The perpendicular distance between those forces is known as the *moment-arm*. The magnitude of $\vec{T}$ is given with $T = moment - arm * F$. In musculoskeletal models we should speak about torques rather than moments. When a muscle that originates at body A exerts a force F_m on an adjoined body B, body B will react with an equal but opposite force $-F_m$(Newton's third law). The joint rotation can be calculated with Euler's equation. The knee-cap and the heel-bone are thought to enhance the torque contribution. Figure 2.2 makes it more clear.
The equations of motion for a free moving rigid in 2D can now be written as:

$$\begin{pmatrix} \sum F_x \\ \sum F_y \\ \sum M \end{pmatrix} = \begin{pmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_{CM} \end{pmatrix} \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{pmatrix} \tag{2.7}$$

If we name the coordinate vector $\vec{q} = (x, y, \theta)$, we can write 2.7 in a more compact form we have:

$$\vec{F} = M\ddot{\vec{q}} \tag{2.8}$$

$\vec{F}$ and M are known as the generalized force vector and generalized mass matrix respectively. In our case $\vec{F}$ is known and our problem becomes:

$$\ddot{\vec{q}} = M^{-1}\vec{F} \tag{2.9}$$

This type of problem is known as *forward dynamics*. We study the motion of a system as a result of known forces acting on it. The other way around is known

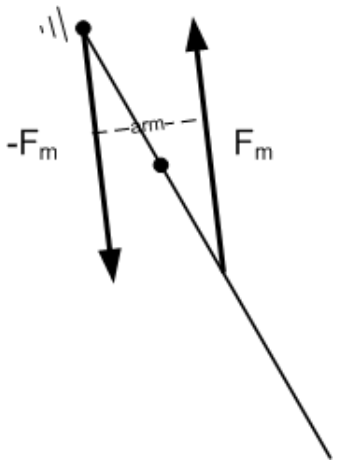


Figure 2.2: A torque example. When force $\vec{F}_m$ acts on a body A that is connected to another body via another body B, body B will react with equal but opposite force $-\vec{F}_m$. We then speak about a torque. arm is the perpendicular distance between those forces and is known as the moment-arm. The magnitude of the torque is given with $T = arm * F_m$. If the torque is not zero the joint will rotate.

as *inverse dynamics*. The aim of inverse dynamics is to calculate the forces and torques for known motions. In mechanical engineering, inverse dynamics is used to solve control problems. The state of the rigid body is determined by $\vec{q}$ and $\dot{\vec{q}}$ and they are our state variables. Our state vector is $\vec{X} = (\vec{q}, \dot{\vec{q}})^T$ and we have:

$$\dot{\vec{X}} = \begin{pmatrix} \dot{\vec{q}} \\ \ddot{\vec{q}} \end{pmatrix} = \begin{pmatrix} \dot{q} \\ M^{-1}\vec{F} \end{pmatrix} \tag{2.10}$$

2.5 Example: free-falling thigh

With the above knowledge we start to model and simulate simple cases. A free-falling thigh for example. The goal is to get familiar with some terms as center of mass, inertia moment etc. and use them. In this case we assume the proximal end of the thigh to be connected with the hip and free at the distal end. Our hip is fixed and only allows planar rotations.

Studying the system

The hip only allows rotation and thus our system has one degree of freedom(DoF). We choose an inertial frame such that its origin coincides with the proximal end of the thigh. See figure 2.1. The body's configuration is given by θ. The other two DoF are constrained by the following equations:

$$\begin{cases} x_{CM} = p\cos\theta \\ y_{CM} = p\sin\theta \end{cases} \tag{2.11}$$

There are two forces acting on the thigh. One is the gravitational force $\vec{F}_g$ and the other is the internal or joint reaction force $\vec{F}_r$. $\vec{F}_r$ ensures the pure rotational

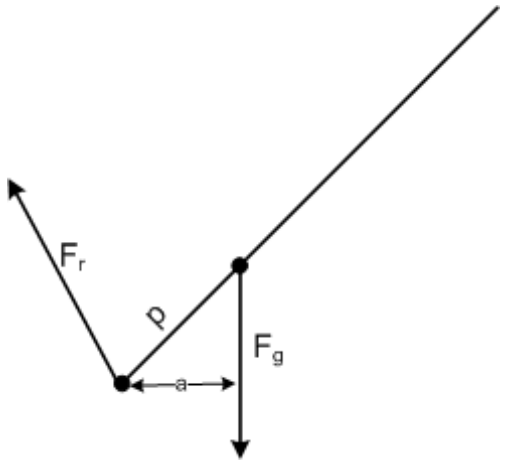


Figure 2.3: The forces that act on the thigh. F_g is known and acts on the CM, along the y-axis. a is the moment-arm of F_g with respect to the hip. F_r is the internal reaction force from the trunk.

motion around the hip and is exerted by the fixed trunk. These forces are drawn in figure 2.3. While $\vec{F_g} = m\vec{g}$ is known, $\vec{F_r}$ should be calculated. Because the hip is fixed and we do not know F_r we write Euler's equation with respect to the proximal end of the thigh:

$$\sum M = I_{hip}\ddot{\theta}$$

By doing this we do not need to know $\vec{F_r}$ explicitly, because its moment contribution with respect to the hip is zero. We have $\sum T = \vec{p} \times \vec{F_g} = mgp\cos\theta$ and our equation of motion is thus:

$$mgp\cos\theta = I_{hip}\ddot{\theta}$$

If we are interested in $\vec{F_r} = (F_{r_x}, F_{r_y})^T$ we can use Newton's equations and equation (2.11) to compute it. Newton's equations are:

$$\begin{cases} \sum F_x = m\ddot{x}_{CM} \\ \sum F_y = m\ddot{y}_{CM} \end{cases} \tag{2.12}$$

With $\sum F_x = F_{r_x}$ and $\sum F_y = F_{r_x} + F_g$ we have:

$$\begin{cases} F_{r_x} = m\ddot{x}_{CM} \\ F_{r_y} = m\ddot{y}_{CM} - mg \end{cases} \tag{2.13}$$

$\ddot{x}_{CM}$ and $\ddot{y}_{CM}$ can be expressed in known terms by differentiating equation (2.11) twice:

$$\begin{cases} \ddot{x}_{CM} = -p(\cos\theta\dot{\theta}^2 + \sin\theta\ddot{\theta}) \\ \ddot{y}_{CM} = p(\cos\theta\ddot{\theta} - \sin\theta\dot{\theta}^2) \end{cases} \tag{2.14}$$

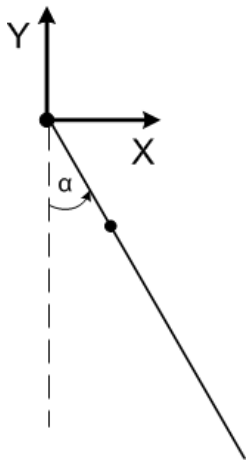


Figure 2.4: A new angle definition. The thigh can be seen as a pendulum driven by gravity. When the thigh is held and released at angle α it will swing back and forth from α to $-\alpha$.

F_r thus depends on $\theta, \dot{\theta}$ and $\ddot{\theta}$. Our system can be seen as a simple pendulum driven by gravity. The behavior of the simple pendulum is well-known. Before we describe the motion we use a different angle definition. We name a new angle coordinate α and is defined to be the angle between the negative y-axis and the thigh(see figure 2.4). When we release the thigh at angle $\alpha = \alpha_{max}$ it will swing back and forth towards the negative direction of the y-axis. Because of conservation of energy the motion will take place between α_{max} and $-\alpha_{max}$. Special situations occur when we release from $\alpha = 0$ degrees or $\alpha = 180$ degrees. The system is then in static equilibrium and will not move.

2.5.1 Implementation with Matlab

The free-falling thigh can be very easily implemented in Matlab. Our state vector $\vec{X} = (\theta, \dot{\theta})$ and its time-derivative:

$$\dot{\vec{X}} = \begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix}' = \begin{pmatrix} \dot{\theta} \\ \frac{p\cos(\theta) f_g}{I_{hip}} \end{pmatrix} \tag{2.15}$$

To start simulation we should specify the simulation time T and the initial conditions $\vec{X}_0$. Implementing the free-falling thigh with Matlab is fairly easy. Below is the Matlab code:

```
options = odeset('RelTol',1e-6);                    % relative tolerance
[T, X] = ode45(@xprime,[0 10], [0, 0],options);     % numerical integrator with
                                                    % simulation time and
                                                    % initial conditions

function dx = xprime(t,x)
    g = -9.81;            % gravitational constant [m/s^2]
    m = 16.944;           % mass [kg]
```

```
p = 0.2100;           % proximal distance [m]
f = m*g;              % gravitational force [N]
mom = f*p*cos(x(1));  % moment around hip [Nm], x(1) = theta
I_cm = 0.4180;        % inertia moment wrt CM [k/m^2]
I_hip = I_cm + m*p*p; % Inertia wrt hip and not Hip_cm [k/m^2]

dx = [x(2); mom/I_hip]; % time derivative of state vector
```

In figure 2.5 we see the results of a simulation for $\vec{X}_0 = (\theta(0), \dot{\theta}(0)) = (0, 0)^T$ and T = 10 seconds. The results agree with our expectations.

2.5.2 Multi-body dynamics

We have successfully simulated the free-falling thigh with Matlab. For more realistic motions we need more bodies however and we enter the field of *multi-body dynamics*. Multi-body dynamics is well studied and can be divided into two areas: $forward$ and $inverse$ dynamics. In inverse dynamics the motions are known and we want to know the forces that cause these motions. In forward dynamics the forces are known and we want to study the motions that are caused by these forces. Our problem can thus be considered as a forward dynamics problem. Currently there are several software packages specialized in solving multi-body dynamics problems. There is even a chip, PhysX, that is specialized in physical computations (www.ageia.com). This chip is mainly used to make games look more realistic. In the next section we will use a multi-body software package called SimMechanics to implement our skeletal models.

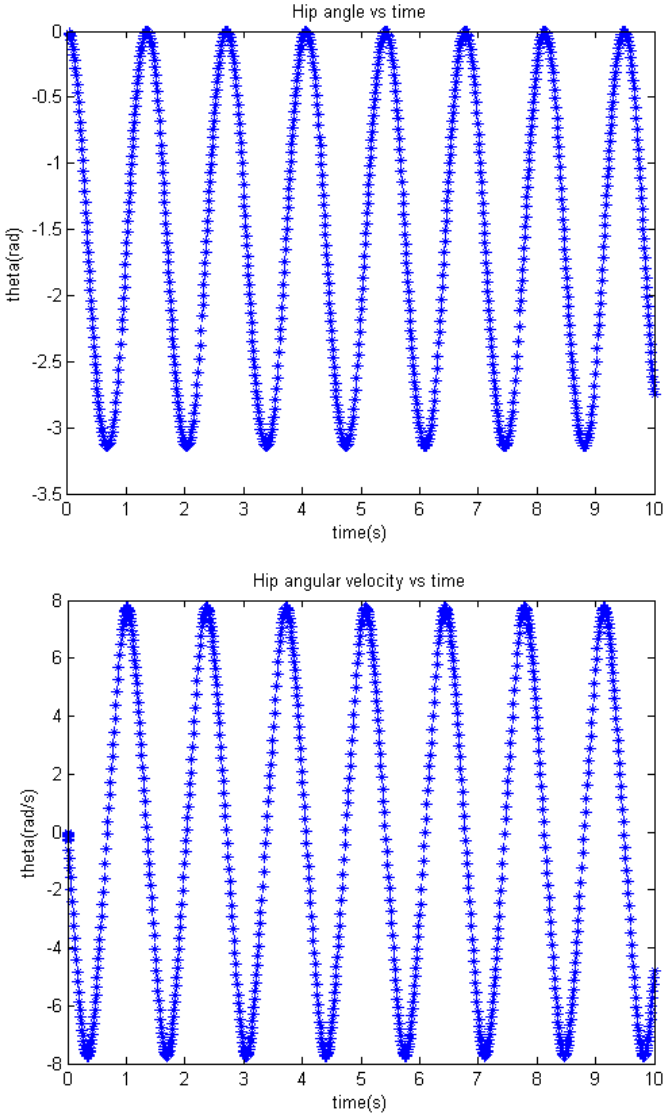


Figure 2.5: Top: hip angle θ as function of time. θ is the angle between the positive x-axis and the thigh. Bottom: hip angular velocity $\dot{\theta}$ as function of time. Initial state: $(\theta(0) = 0, \dot{\theta}(0) = 0)$. Because we have a free rotational case and no loss of energy θ will go from 0 rad to $-\pi$ rad back and forth.

2.6 Modelling with SimMechanics

In the previous section we have implemented the free-falling thigh with Matlab. Although it only took a few lines of code, things become more complex if we want to add more features and/or bodies. Another way to study the dynamics of multi-body system is the use of dedicated software packages. Such a package is SimMechanics. In [7] SimMechanics is chosen to be the best multi-body systems software for integration with a muscle software package called MMS. More information about SimMechanics can found at http://www.mathworks.com/products/simmechanics.

2.6.1 What is SimMechanics and why

The way of modelling with SimMechanics differs from the way we are used to. SimMechanics uses blocks that represent idealized physical objects like rigid bodies, revolute joints etc. With these blocks we can build fairly complex physical systems [6]. SimMechanics can automatically formulate the equations of motions. SimMechanics requires Simulink and Matlab to be installed.

2.7 Free-falling thigh with SimMechanics

2.7.1 Building the model

In section 2.5.1 we have modelled a free falling thigh with the Matlab-code. We will use SimMechanics to model the free-falling thigh again with same data. The steps are explained one by one.

1. Specify the machine environment. With the environment block we model the environment for our system. Gravity, the motion's dimension(2D) and analysis mode(forward dynamics) are specified here. Error-tolerances for several cases can also be set. We use the default values.

2. Choose an inertial frame. The ground is the ever-present body in our models. We attach a frame to the ground or some object that is fixed to the ground. This frame is our inertial frame. We as observers are particulary interested in the output(motion and configuration) expressed in this frame. The origin of the frame is set to [0,0,0] and it coincides with the hip. Because the hip is fixed to the trunk, we name our ground block trunk.

3. Add a revolute joint. If we want to add another body to an existing body we should use joints. If we add an airborne body in space, we can use a joint with 6 DoF. We want the thigh to rotate in the XY-plane. This is done via a revolute joint with axis of action [0 0 1]. The revolute joint models the hip.

4. Add the thigh. We will model the thigh with a rigid body. SimMechanics's body block represents a rigid body. In the body block we should fill in its mass and inertia-tensor [2] with respect to the center of mass. Then the

[2]The inertia tensor is a 3x3 matrix that specifies resistance of the body against rotation. In 2D there is only rotation around one axis in de Z-direction. Therefore we only need to fill in I_{33}.

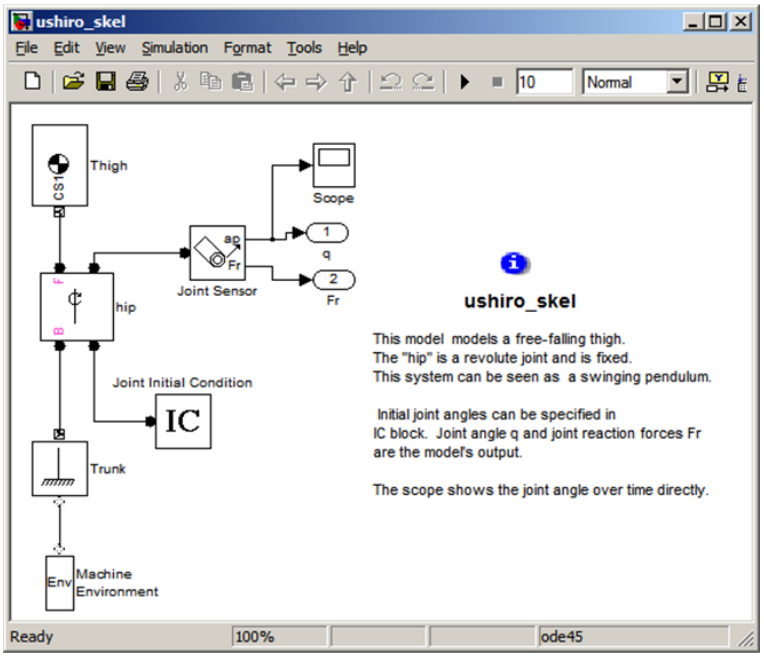


Figure 2.6: The thigh model in Simulink, built with SimMechanics. It is a block diagram. A SimMechanics block models a physical object. The joint sensor block for example can sense different quantities like joint angle, joint angular velocity, joint reaction force etc.

configuration of the body should be specified. With 3 coordinate systems we can specify the main points of the body. They are: the center of mass, the proximal end and the distal end. The specifications can be done in many ways. Once the position of the origins of the frames are set, we should also specify their orientations. The output of our revolute joint is the difference of the orientation of the ground frame and the proximal frame. In our case, we can let the orientation of the CS-es to be default. In the 3D case the orientation of the CS of the center of mass depends on how the inertia tensor is given.

5. Add a joint sensor. The joint sensor can sense joint motions and joint reaction forces.

2.7.2 Simulation

Now that our model is built we can start the simulation. First we need to specify the initial condition for our joint. The zero state of the joint is determined in the specification for the thigh's CS's. During simulation numerical integration should be done. SimMechanics can be fully used with Simulink and we can use its implemented numerical solvers directly. Output can be generated via Simulink's output blocks. With SimMechanics we can literally view our output via animation. Animating the system's motion can significantly slow down the simulation however. Then we should fill in the simulation time. Simulation is started by pressing the play button. All these are done via the GUI of SimMechanics. For serious experiments it is easier to run the model programmatically. With Matlab we can write scripts that simulates the model under different circumstances.

Free-falling thigh

We named our model ushiro_skel. The solver is set to ODE 45 with relative tolerance $reltol = 10^{-6}$. We have chosen θ to be a joint coordinate. The initial condition is $\theta = 90$ which means that the thigh is parallel to the ground. The simulation time is set to 10 seconds. Simulating the model with Matlab' command line goes with:

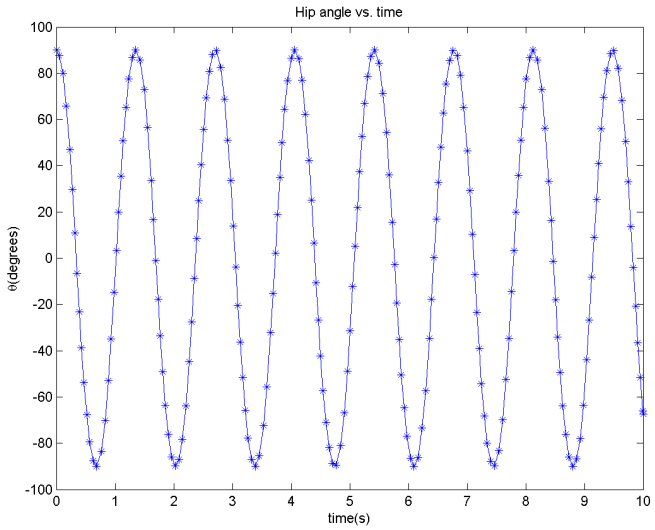
```
sim('ushiro_skel');
```

In figure 2.7 we see the output of the model. This time with joint reaction forces. Assigning the model's output to a variable y is done with:

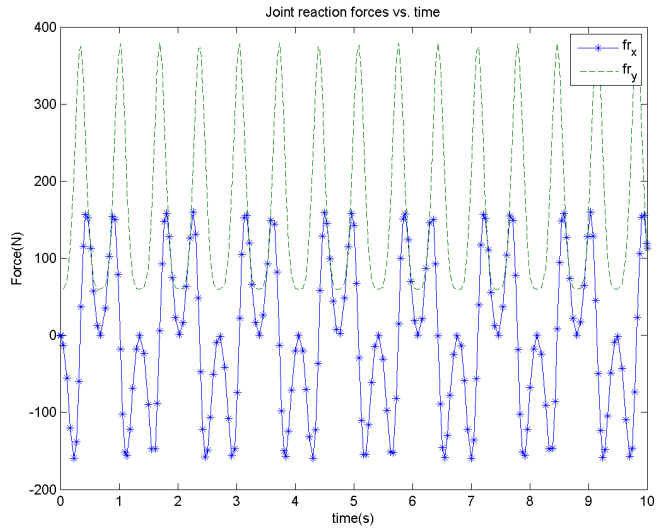
```
[t,x,y] = sim('ushiro_skel')
```

Here t is the time vector, x the state matrix, and y the output matrix. The dimension of y equals $nt \times n$, where nt is the number of time-steps and n is the number of output variables. So each column i of y contains the value of variable i over time. When using variable time-step solvers the time-steps will vary. Sometimes we want to have values at specific time-steps. We can specify an output time-vector. The output values are interpolated.

```
T = [t_start:dt:t_end]
[t,x,y] = sim('ushiro_skel',T)
```



(a)



(b)

Figure 2.7: (a) The hip angle. It agrees with what we have already found, but this time the angle definition is different. (b) The joint reaction forces that the thigh will feel. The formulation is given in equation (2.13). These forces ensure the rotational motion. Due to gravity that acts in the y-direction F_{r_y} is larger than F_{r_x}. With mass $m_{thigh} = 16.944kg$ the gravity force equals 166.22N. The y-component reaches peaks that are much larger than the gravity force. Joint reaction forces are interesting for those who want to study injuries.

id	Name	Length l(m)	p(m)	Mass m(kg)	Inertia I_{CM} (kg*m*m)
1	Thigh	0.485	0.210	16.944	0.418
2	Shin	0.458	0.198	7.074	0.136
3	Foot	0.165	0.045	2.468	0.020

Table 2.1: Some realistic segment parameters.

id	Name	$\theta_{min}(deg)$	θ_{max}(deg)	Stiffness	Damping
1	Hip	-5	110	100	40
2	Knee	-150	0	1000	40
3	Ankle	5	45	1000	40

Table 2.2: Joint parameters. The extreme angles are given with respect to the joint coordinates in a right-handed coordinate frame. The zero joint state is defined as the state where both bodies are aligned.

We can also change the solver and its settings. The default solver is ODE45 with relative tolerance $relTol = 10^{-3}$. Changing the solver with other relative tolerance goes with:

```
myopt = simset('Solver', 'ode23','RelTol',1e-3); %other solver + reltol
[t,x,y] = sim('ushiro_skel', T, myopt);
```

2.8 Skeletal data

To build a skeletal model we should use realistic data for the body segments and joints. These data are obtained via special measuring techniques. With [8] we can calculate some average skeletal parameters. This source only provide skeletal data however. For our musculoskeletal model we need muscle data as well. These muscle data should fit with the skeletal data and we thus have a problem. Luckily, the faculty of Human Movement Sciences of de Vrij Universiteit of Amsterdam has provided us some realistic musculoskeletal data. The skeletal data are listed in 2.1 and table 2.2.

2.8.1 Free-falling thigh with realistic hip.

In real-life the body joints and thus also the hip allow a limited rotation range. Ligaments and anatomical structure of the skeleton are responsible for this. We model this by adding stiffness and damping to the joint if the joint angle exceeds one of the two extreme angles. The damping and stiffness will produce a counter-moment that is proportional to $|\theta - \theta_{extreme}|$. The proportion is given with damping-constant k_1 and stiffness-constant k_2 respectively. We have:

$$damping = \begin{cases} -k_1(\theta - \theta_{min}) & if \quad \theta < \theta_{min} \\ -k_1(\theta - \theta_{max}) & if \quad \theta > \theta_{max} \end{cases} \tag{2.16}$$

$$stiffness = \begin{cases} -k_2(\theta - \theta_{min}) & if \quad \theta < \theta_{min} \\ -k_2(\theta - \theta_{max}) & if \quad \theta > \theta_{max} \end{cases} \tag{2.17}$$

For the hip we have:

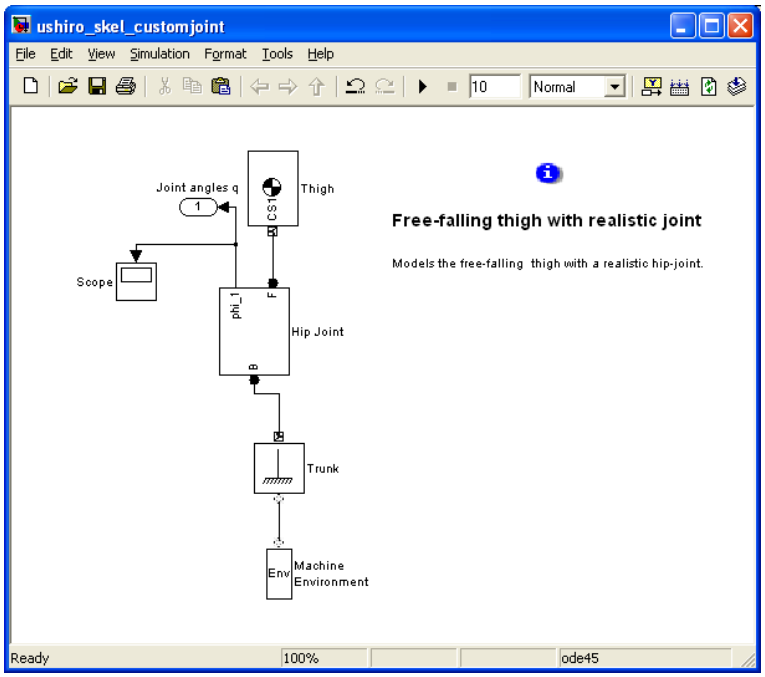


Figure 2.8: The new free-falling thigh model. Here a realistic hip is added.

- θ_{min} = - 5 deg
- θ_{max} = 110 deg
- damping-constant = $100\frac{Nm}{deg}$
- stiffness-constant = 40 $\frac{Nm}{deg}$

In figure 2.8 we see the block-diagram. Things become more interesting if we open the hip block in figure 2.9. In figure 2.10 we see the motion of the our free-falling thigh. With $\theta_0 = 90$ degrees, $T = 10$ seconds. This time the motion does not go from 90 to -90 degrees.

2.9 Conclusion

With SimMechanics we can very easily build a skeletal model and study its motion with known forces acting on it. This is known as forward dynamics. With forces we mean both rotational(torques) and translational forces. We have built a skeletal model that models the thigh with a realistic hip. Extending the model with more bodies is very easy. The only forces that act on this model are gravity and force produced by ligaments. In mechanical systems joint actuators are placed at the joints to drive the systems. These actuators can produce any force at certain time if it is within their limit. In the musculoskeletal system skeletal muscles are the joint actuators. Their force-production depends on many factors. Controlling the musculoskeletal system is therefore more complex than controlling mechanical systems. Luckily we have a very powerful controller

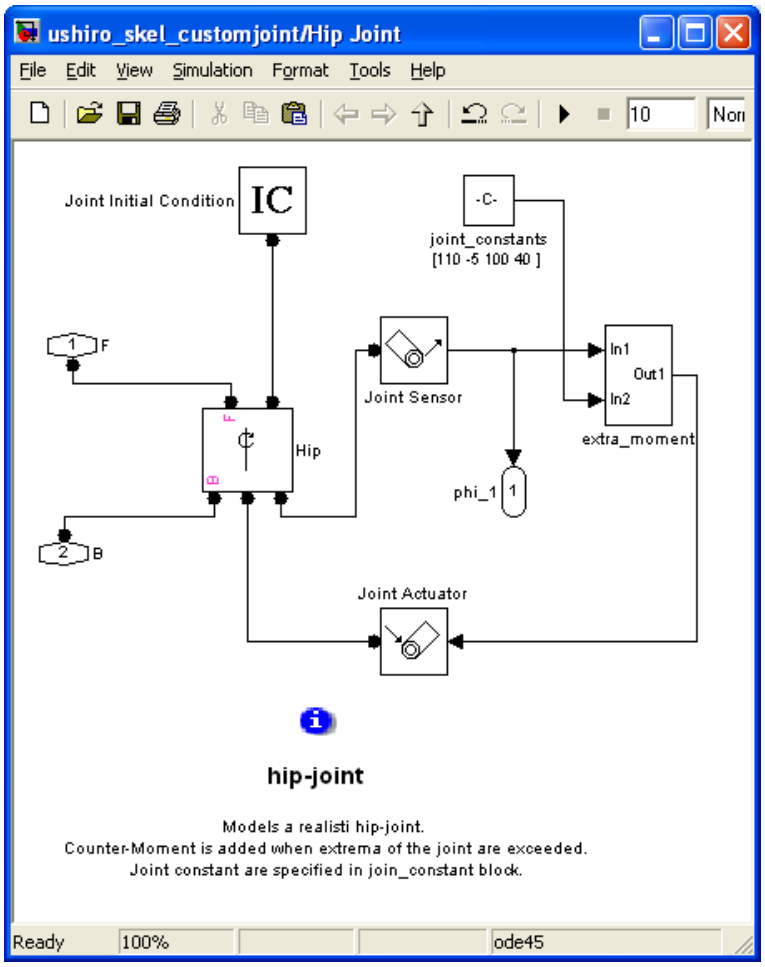


Figure 2.9: The more realistic hip joint. The joint sensor senses the joint angle. In the extra-moment block we calculate the contribution of the hip-ligaments. With the joint actuator we can actuate the hip with this contribution.

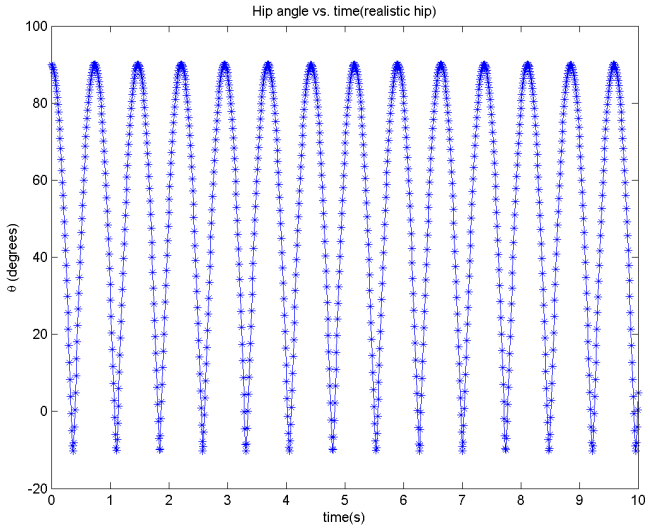


Figure 2.10: The motion of the free-falling thigh with realistic hip, started from a horizontal position. The minimum angle is set to -5. From the plot we can see that it is not a hard limit. When the minimum angle is exceeded counter-moment is added. This can be seen as an elastic collision.

in real life and that is the human mind. In the next chapter we will build skeletal-muscles that produces torques for the skeletal model.

Chapter 3

Building skeletal-muscle models.

The skeletal-muscle is a complex biological structure. Its main function is to generate force through contraction. The force is transferred to the skeleton via tendon and aponeurosis, that connect the muscle with the skeleton. This force allows us to move but also protects our bones against impact forces. Bones are relatively weak against bending forces compared with compressing forces. By contracting the right muscles the bending force can be partly or mostly neutralized. We are mainly interested in the force production behavior of the muscle. In this chapter we first look at the structure of the skeletal muscle and how it contracts. Then we look at how muscle are modelled mathematically. After that we build a thigh muscle model and add this to our thigh-skeletal model.

3.1 Structure of skeletal muscles

Skeletal muscles are connected to the skeletal bones via tendons on both ends. The proximal attachment point and the distal attachment point are named *origin* and *insertion* respectively. Different connective tissue sheaths having their origins on both tendons run from end to end through the muscle, subdividing the muscle in units, subunits till the muscle cell or fibers. These sheaths are thus continuous and prevent the muscle from pulling out during contractions. The largest unit is the skeletal muscle itself. It is surrounded by a fascia and a connective tissue sheath called epimysium(epi = above and my = muscle). Then the muscle is subdivided into muscle bundles or fascicles by a connective tissue sheath called perimysium (peri= around). Within a fascicle we have the muscle fibers or muscle cells, kept away from each other by endomysium (endo = internal). Each muscle fiber is surrounded by a delicate membrane called the sarcolemma. Unlike other cells, the muscle cell is shaped long. If we want to zoom in, we have to use a light microscope that can reach a magnification of 1000 times the objective. Under the microscope we will see that a muscle fiber is made up of parallel myofibrils. In figure 3.1 we can see the structure of a skeleton muscle. A remarkable observation is the striated pattern that can be seen. A myofibril is made up with small contractile units in series called sarcom-

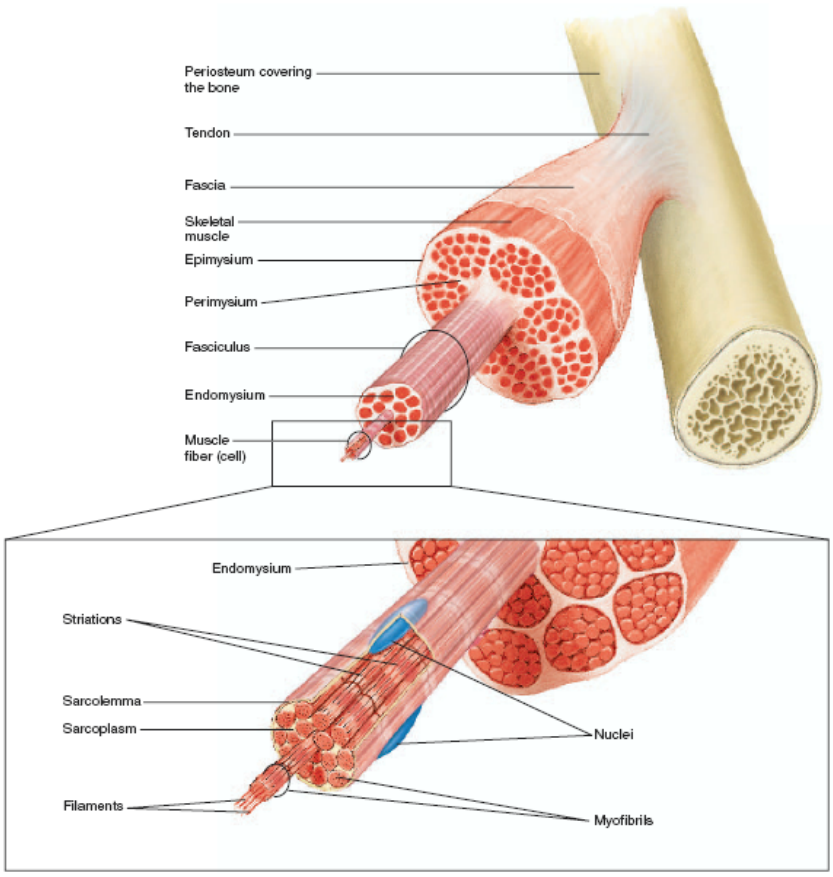


Figure 3.1: The structure of a skeletal muscle. The muscular part is grouped into muscle bundles or fascicles. Each fascicle contains a number of muscle cells or fibers. A muscle fiber is made up of parallel myofibrils. The myofibril in turn is made up of sarcomeres in series. Image taken from [13].

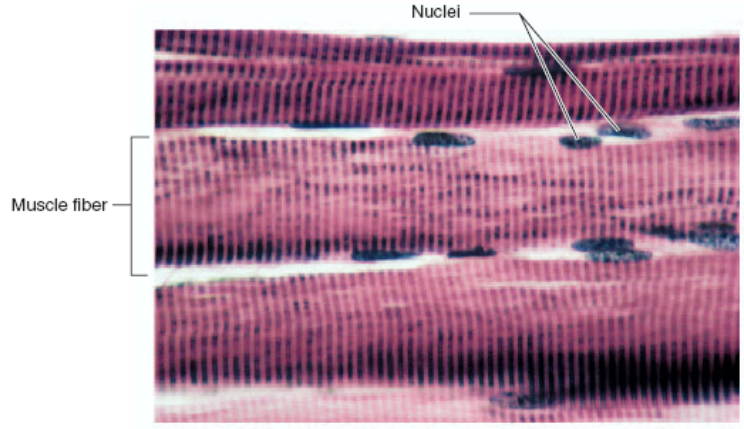


Figure 3.2: Skeletal muscle fibers under a light-microscope. We can notice a striated pattern. The white rings are formed by actine filaments, while the myosin filaments form the darker rings. The nucleus is an organelle that contains the fiber's genetic information. Image taken from [13].

eres. Each sarcomere is built up with thick and thin filaments, called myosin and actin filaments respectively. Myosin filament has a dark color, while actine filament has a lighter color. Both filaments lie parallel to each other with partial overlapping, resulting in a striated pattern(see figure 3.2). Skeletal muscles are also called *striated muscles*. In other type of muscles the sarcomeres are not aligned in series. These muscles are known as *smooth muscles*. Examples of smooth muscles are heart and kidneys. Furthermore a sarcomere is bordered with Z-lines. [1]

3.2 Activation of the skeletal muscles

Skeletal muscles are voluntary muscles. We can activate them at will. The smallest unit that we can activate individually is the fascicle. A fascicle contains only one type of fiber. There are two main types of muscle fibers, slow-twitch and fast-twitch. They differ in the way they perform their metabolism. Slow-twitch fibers have relatively large endurance, whilst fast-twitch fibers are not. Each fascicle is controlled by a α-motoneuron that is located in the spinal cord. A fascicle with its motoneuron is called a motor unit. The motoneuron innervates all the muscle fibers of its fascicle with an action potential. The larger the fascicle the larger the action potential. [2] As a result each muscle fiber will fire a new action potential that will travel along their length(see figure 3.3). This will result into the release of Ca^{++}ions. With the release of Ca^{++} ions,

[1] Muscle strain is something that we are all familiar with. It indicates broken Z-lines. Muscle strain is usually the result of eccentric contraction. In eccentric contractions the muscle force is overcome by a larger force and will lengthen instead of shorten. If the difference is too big, microscopic tears will take place in de Z-lines.

[2] An action potential is a local voltage change that travels along the membrane of a cell.

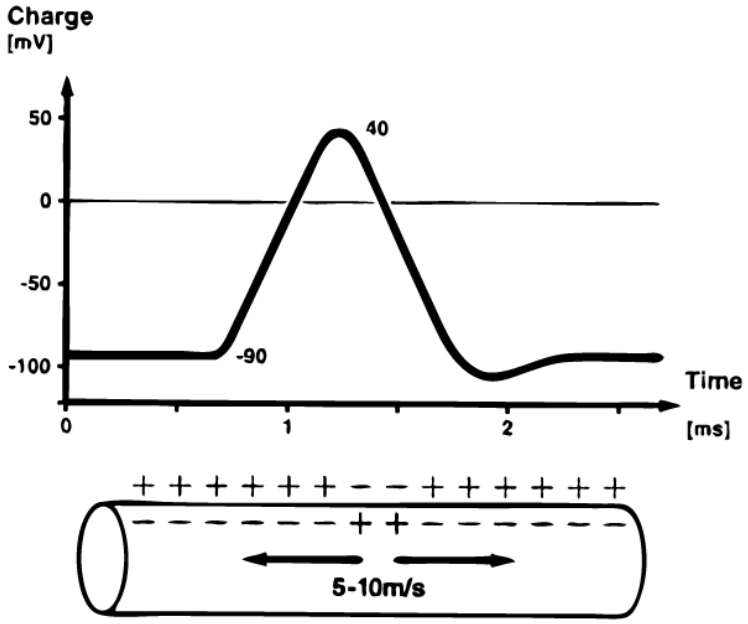


Figure 3.3: The action-potential of a muscle fiber(top) and how it is propagated through the fibers(bottom). Notice that the action potential starts from the middle of the muscle fiber where the junction of the fiber and motoneuron is formed. Image taken from [9].

cross-bridges can be formed and the fibers will contract. The force result from an action potential is called the *twitch* force. Slow-twitch fibers produces slower twitch force than fast-twitch fibers. The fire frequency is the number of action potentials per second. The force production depends on the fire frequency. The sum of electrical activation of the fascicle can be measured with electromyography(EMG). The output is called a myogram and depends on the number of activated fibers, fiber-type, motor-unit composition and many more. The amplitude of the signal corresponds with the activation level. A fascicle has an activation threshold. It will only be activated if the activation level exceeds this threshold. The threshold depends on the fiber type and is proportionate with the size of the fascicle(=number of muscle fibers) [3]. The process of how fascicles get activated is called recruitment. Slow-twitch fibers have lower activation thresholds and fascicles with slow-twitch fibers are usually smaller than those with fast-twitch fibers. In slow movements like typing/walking the fast-twitch fibers are not activated. They are only activated when explosive movements are needed. This can be explained with the economic metabolism of the slow-twitch fibers.

3.2.1 The cross-bridge theory

The basic theory behind muscle contraction is the cross-bridge theory(Huxley, 1957). For a basic explanation, we zoom in at sarcomere the level. There,

[3]This is known as the Henneman size principle

sarcomere is the basic contraction unit and has an optimal length of around $2.7\mu m$. It is made up with actine and myosin filaments, named after the main protein molecules they are build with, and titin filaments. Actine and myosin filaments are also named as thing and thick filament respectively, according to their diameter. A cross-bridge is a temporary bound between thick and thin filaments. Once bound it will pull the thin filaments towards the center of the sarcomere and let loose. The titin filaments keep the thick filaments in the center of the molecule. The number of cross-bridges that can be formed depends on several factors:

The length of the sarcomere: the cross-bridges can only be formed if there is overlap between the thick and thin filaments. The cross-bridge theory assumes that cross-bridges are uniformly distributed and therefore the force scales linearly with the amount of overlap.

Change in length of the sarcomere: the rate of forming cross-bridges depends on wether the sarcomere lengthens or shortens.

Active state of the sarcomere: binding sites for the cross-bridges are only freed if the concentration $[Ca^{++}]$ exceeds a certain threshold. During activation calcium-ions are released. When activation stops, the calcium ions are actively pumped back.

3.3 Brief overview of current models.

Current muscle models can be divided into 3 categories. Namely:

Huxley: This model keeps track of the state of a finite number of cross-bridges. Furthermore the transition of these states over time is described by a set of equations. It needs lot of computation power and lots of equation coefficients. Theoretically this type of modelling has the highest accuracy.

Distribution Model(DM): In order to decrease the computational effort, the DM model groups the individual cross-bridges into several groups. The behavior of the groups is described.

Lumped Element Model: These models are composed of force producing elements from the world of engineering. The elements can be connected to each other either parallel or serial. Hill's muscle model is the most well-known lumped element model. Most muscle models currently used are of this category.

The muscle model that we will use falls in the category Lumped Element Model.

3.4 Muscle experiments

Unlike in mechanics we do not have theories that describes the complex muscle behavior well. We have to rely on muscle experiments. Once these experiments are done we can formulate equations and specify coefficients that describe the muscle's behavior. The most well-known muscle experiments are listed below:

isometric The length of the muscle is kept constant and the muscle is maximally activated. The maximal produced force is that length is called the *tetanic* force.

isotonic A known constant force is put on the muscle and quantities of interest are measured.

isokinetic The contraction speed is kept constant and relevant quantities are measured.

To get accurate data from the muscle of interest, it should be isolated from other muscles. Dependent on which state the experiment is done the muscle is cut away from its neighboring muscles. The experimented muscle can be in different states.

in vivo Latin for "(within)" in the living". The experiment is done on the living tissue of a whole, living organism.

ex vivo Latin for " out of the living)". The tissue is still alive but outside in the organism.

in vitro Latin for "(with)in the glass". These experiments are done in a controlled environment, where the biological structure is separated from its owner.

in situ Latin for " in the place". The experiment is done on the normal operation location of the tissue. The owner could be dead or alive.

It becomes clear that some types of experiments cannot be done on human muscles. Therefore animals are used. When performing such experiments scientists try to keep the number of animals used as low as possible.

3.5 Virtual Muscle

Virtual Muscle 3.1.5 is a program that builds muscle-tendon complexes and describes their behavior. It is written in Matlab and the muscle models can be exported as Simulink blocks. In the Simulink environment these block can be fully integrated with other Simulink blocks. The equations that Virtual Muscle uses to describe the behavior of skeletal muscles are determined by best-fit procedures on data that come from many experiments. These experiments studied the muscle behavior under a wide range of physiological conditions. For people who are interested in how this is precisely done we refer to [16][4]. Hopefully the behavior can be extrapolated and we get realistic results.

3.5.1 Hill's muscle model

The Virtual Muscle Model is a Hill's based muscle model and is categorized as a lumped muscle model. It treats the fascicle as a scaled version of a sarcomere. It assumes that all sarcomeres behave uniformly within a muscle fiber and that all fibers within a fascicle also behave the same. System of engineering alike elements are then used to represent the muscle-tendon complex (*mtc*). To understand how Virtual Muscle works it is better to look at the more simple Hill muscle model first. In Hill's muscle model the tendon is modelled with a *passive* elastic force element, whilst the muscle group is modelled with a *passive* and an *active* force element. The force of the active element CE depends on its length L_{CE} and neural activation *act*. The force of the passive elements PE and SE depends on their lengths. Where $l_{PE} = l_{CE}$ and $l_{SE} = l_{OI} - l_{CE}$ and l_{OI} [5] is the mtc-length. In figure 3.4 we see a scheme of the Hill's model. The *characteristics* of the force-producing elements are:

Passive element SE : $F_{SE}(l_{OI} - l_{CE}) = k_{SE}(L - L_{slack})$ if $(L \geq L_{slack})$ (half parabola). k_{SE} is the stiffness constant of the tendon. Experiments have shown that the tendon is very adaptive. We have $l_{SE,slack} \approx 0.96 L_{SE,opt}$, where $L_{SE,opt}$ is the length of the tendon when maximal muscle force is applied to the tendon.

Passive element PE : $F_{PE}(l_{CE}) = k_{PE}(L - L_{slack})$ if $(L \geq L_{slack})$ (half parabola). k_{PE} is the stiffness constant and is determined experimentally.

Active element CE : this the most challenging element and 3 relations are used to model the behavior, we have

- **Isometric Force-length** $F_l(l_{CE})$**:** $F_{max}(1 - (\frac{L_{CE} - L_{CE,opt}}{W})^2)$. This is a parabolic curve and W is a measure for the width of the parabola. This relation agrees with the cross-bridge theory.
- **Isokinetic Force-velocity** $F_v(\dot{l}_{CE}$**:** $\frac{F_{max} b - a v}{b + v}$. This relation is noticed by Hill in 1938, and the constants a and b are fiber-type dependent.
- **Active State** a_f**:** $\dot{a_f} = \frac{1}{\tau}(stim(t)$ The active state can be seen as the concentration of bounded Ca^{++} ions. This equation models the relation between the stimulation and active state in a very simplified

[4] People with a weak stomach are not recommended to read this. Many cats were sent to heaven...

[5] O and I stand for origin-point and insertion-point of the mtc. The origin point is "closer" to the body than the insertion point.

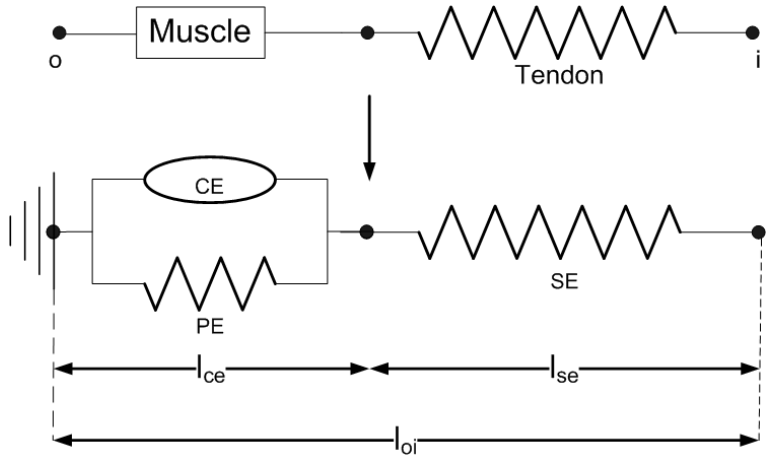


Figure 3.4: Schematic view of the Hill's muscle model. With Hill's model we want to predict the force behavior of the muscle-tendon complex(mtc), with inputs neural activation and mtc-length(l_{OI}). The muscular part is modelled with two elements: a parallel elastic element PE and a contractile element CE. The tendon part is modelled with a serial elastic element SE. The force of the SE is the output of the muscle model.

way. The active state is very difficult to model, because it depends on many factors(bio-chemical, bio-electrical etc). Hill based muscle models differ the most in how they model a_f. Virtual Muscle for example uses a set of coupled ODEs to model this relation.

The algorithm for Hill's model

In algorithm 1 we see how we can simulate Hill's muscle model.

Input: Neural activation function $act(t)$ and mtc-length $l_{OI}(t)$
Output: Tendon force F_{SE}
for *t=0:dt:T* **do**
  $F_{SE} = f_{SE}(l_{OI} - x)$;
  $F_{PE} = f_{PE}(x)$;
  $F_{CE} = F_{max} * f_{l,CE}(x) * f_{v,CE}(\dot{x}) * a_f$, where ;
    $\dot{a}_f = f_{af}(act)$;
      $a_f = a_f + \dot{a}_f$;
  $\ddot{x} = \frac{F_{SE}-(F_{PE}+F_{CE})}{m}$;
  $\dot{x} = \dot{x} + dt\ddot{x}$;
  $x = x + dt\dot{x}$;
end

Algorithm 1: Algorithm for Hill's muscle model. A stabilization mass m is used to keep track of the length of CE. The position of the mass is given with x and tells us what the length of CE is. In the algorithm $f_{l,CE}$ and $f_{v,CE}$ functions are the normalized force-length and force-velocity relationships with respect to F_{max}. Furthermore f_{af} is normalized to [0,1]. If we want to know the real F_{CE} we should scale $f_{l,CE} * f_{v,CE} * a_f$ with F_{max}. We used Euler method to solve x, $\dot{x}$ and a_f for easier understanding.

3.5.2 The Virtual Muscle model

Hill's model only has good prediction for maximal activation. Most human motions occur at *sub-maximal* activation levels. Virtual Muscle is developed by the *Alfred E. Mann Institute*, California and its purpose is to provide an accurate muscle model. Virtual muscle is a Hill's based muscle model, where the force-characteristics of the different elements are obtained through many (animal) experiments. The results are different characteristics for the elements. Especially the active state a_f has great differences with the a_f of Hill's model. Observed phenomena are(rise time, fall time, sag, yielding) used to describe a_f rather than chemical reactions. Virtual muscle model is also supposed to give good force prediction at sub-maximal activation levels. To do this the muscle group is divided into multiple units. In algorithm 2 we see the algorithm for Virtual Muscle. For a precise formulation of the various relations we refer to [10].

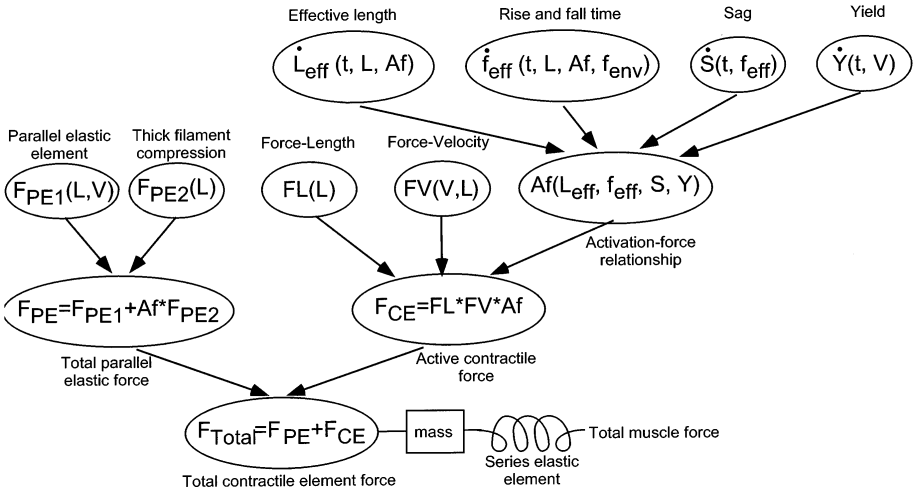


Figure 3.5: Diagram of relations that Virtual Muscle uses to model a muscle unit. Only the diagram of one unit is given. A muscle unit has multiple state variables, that should be solved. The more units are modelled, the more state variables are needed. The sum of the forces of all units forms the total contractile force. This force will be transferred to the skeleton via the SE(tendon) element. A mass is used to keep track of the length of the muscle group.

Input: Neural activation function $act(t)$ and mtc-length $l_{OI}(t)$
Output: Tendon force F_{SE}
for *t=0:dt:T* **do**
$F_{SE} = f_{SE}(l_{OI} - x)$;
$[stim] = f_{recuit}(act(t))$;
$F_{muscle} = \sum_{i=1}^{n} F_{unit,i}(x, \dot{x}, stim_i)$ where ;
$F_{unit,i} = f_{PE,i}(x) + f_{CE,i}(x, \dot{x}, stim_i)$;
$\ddot{x} = \frac{F_{SE}-(F_{muscle})}{m}$;
$\dot{x} = \dot{x} + dt\ddot{x}$;
$x = x + dt\dot{x}$;
end

Algorithm 2: Algorithm that Virtual Muscle uses if more units are modelled and sub-maximal activation is considered. During sub-maximal activation not all muscle-units or fascicles are recruited. The function $f_{recruit}$ turns the activation into different stimulation frequencies for the different units. f_{CE} is a very complex function and a system of coupled ODEs should be solved to obtain f_{CE}. The more units we use the more time the simulation will cost.

In figure 3.5 we see a schematic view of the relationships between the different elements. In figure 3.6 we see the way Virtual Muscle models a mtc.

3.5.3 The model's variables

Input variables: the model has 2 input variables.

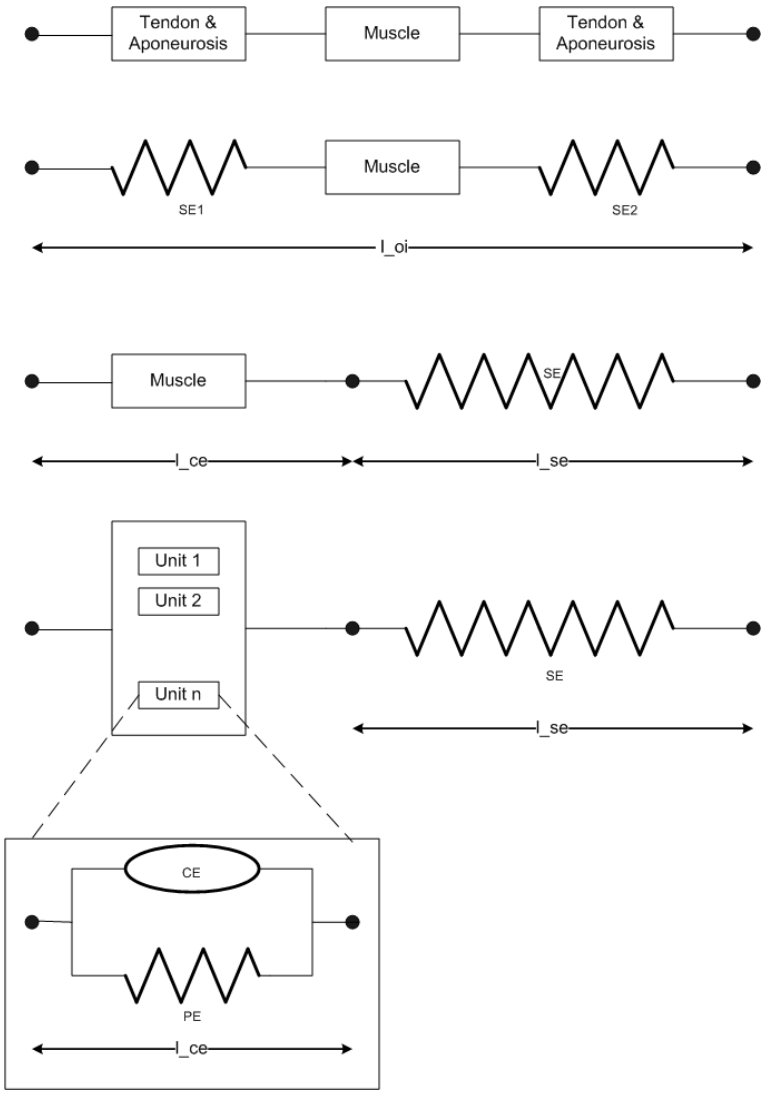


Figure 3.6: Schematic overview of how muscle-tendon complexes are modelled with Virtual Muscle. Here the muscular part is divided into multiple units. Each unit is modelled with a CE and PE element. During sub-maximal activation not all muscle units are active. Smaller units have lower activation thresholds than bigger units.

- Neural activation *act*: it should have a values between 0 and 1. 0 means no activation, all the fibers are at rest. 1 means full activation, all the fibers fires action potentials at their maximum frequencies. With this value the effective firing frequency for each fascicle is calculated. This models the recruitment. The result is scaled to $f_{0.5}$. $f_{0.5}$ is the firing frequency where half of the isometric force is reached. Each fascicle has its own $f_{0.5}$. Recorded data from EMG could be used as input. The data should be scaled to the level of maximal voluntary contraction. Another source is data from a simulated α-motoneuron. $act(t)$ is also the input for our musculoskeletal model. In our optimization process we typically want to find $act(t)$.
- Length l_{oi}: the length of mtc and depends on the skeletal configuration. Because the skeletal model outputs joint coordinates q, the $l_{oi}(q)$ should be calculated.

The state variables: The state of the whole muscle model is determined by the length of the fascicles l_{CE} and a set of variables that determines the active state of the fascicles. The model uses 4 variables to specify the state of a fascicle.

The output variables: The output is the force of the tendon F_{SE} in Newtons and has a positive sign. If we want to couple the muscle models with the skeletal model we should take this into account.

3.6 Using Virtual Muscle 3.1.5

Virtual Muscle 3.1.5 comes with two programs:

BuildFiberTypes: With BuildFiberTypes we can build fibers by specifying their specific coefficients. The relations are normalized to maximal force F_{max} and optimal state variables. Because fibers can be scaled to fascicle level, this program can be seen as BuildFascicles.

BuildMuscles: with BuildMuscles we can build mtc-models and export them as Simulink blocks. Data for the mtc we want to model are needed.

3.6.1 Building the fibers

The muscle-fiber data we used came with Virtual Muscle 3.0. It contains coefficients for super-slow(SS), slow(S) and fast(F) human muscle fibers. The coefficients are obtained after lots of real muscle experiments and should be normalized to optimal state variables. Virtual Muscle treats the muscle fiber as scaled sarcomere. Therefore it uses normalized force-state relation to describe the fibers behavior. If we know the force-state properties of a muscle fiber, then the force-state properties of the fascicle is the product of the number of muscle fibers. A measure for the number of muscle fibers is the physical cross-sectional area(PCSA). The PCSA is given with $PCSA_{fasc} = \frac{Volume_{fasc}}{length_{fasc}}$. When we know the optimal length of a fascicle and its PCSA we can calculate its optimal force with a constant called *specific tension*(N/cm^2). The *specific tension* for human muscle fibers is around 31.8 $\frac{N}{cm^2}$ and is experimentally determined. For humans

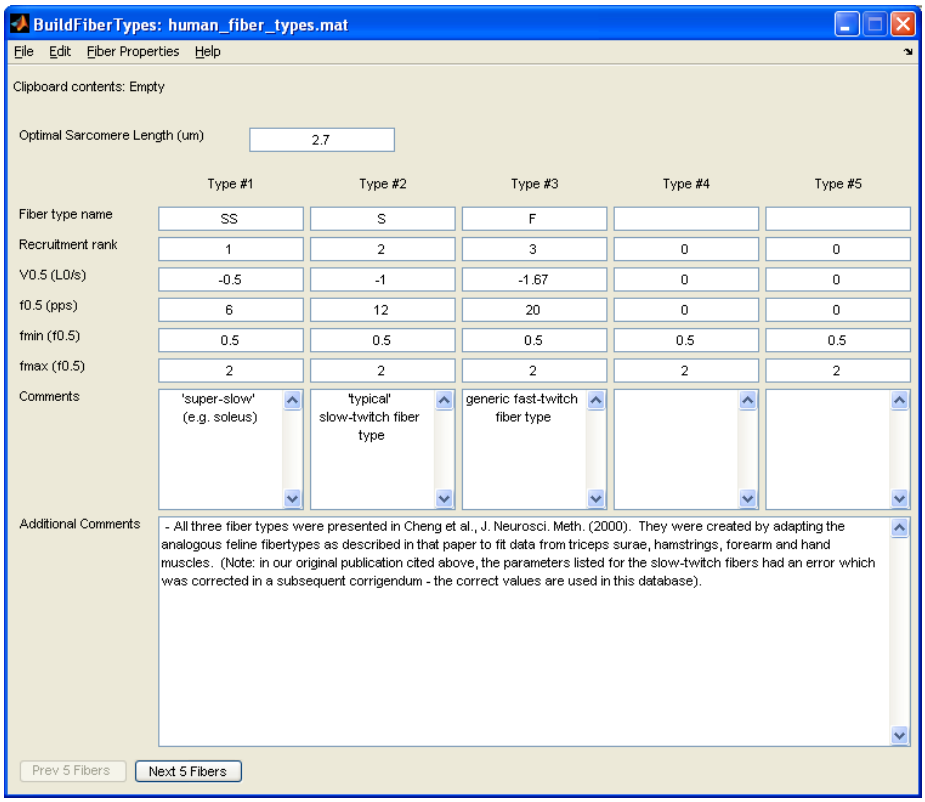


Figure 3.7: With BuildFiberTypes program we can build muscle fibers by specifying different coefficients. The behavior of the muscle fibers are normalized in optimal force and optimal state variables. A fascicle is treated as a scaled fiber.

the optimal fascicle length corresponds with its resting length. So in a relaxed state the muscle has the greatest force production potential [6].

3.6.2 Building the muscles

With the BuildMuscles program we can build mtc-models by specifying a number of parameters. The parameters that scale the normalized force-state relations to real values are:

Muscle mass m(g): the mass of the muscle.

Optimal fascicle length L_o (cm): average length of the muscle belly to produce optimal isometric tetanic force.

Physical cross sectional area PCSA (cm^2): can be seen as a measure for the amount of muscle fibers. It is automatically calculated via $PCSA = \frac{V_{muscle}}{L_o} = \frac{m\rho_{muscle}}{L_o}$. ρ_{muscle} is the muscle density and is assumed to be $1.06\frac{g}{cm^3}$

[6] The starting positions of sprinters and weightlifters make use of this knowledge.

id	Name	Function	L_0(cm)	$l_{0,T}$(cm)	F_o(N)
1	Biceps femoris	hip extensor, knee flexor	10.4	37.0	4000
2	Gluteus maximus	hip extensor	20.0	15.0	5000
3	Rectos femoris	hip flexor, knee extensor	8.1	34	4500
4	Vastii	knee extensor	9.3	16	13500
5	Gastrocnemius	knee flexor, ankle extensor	5.5	37.6	4000
6	Soleus	ankle extensor	5.5	23.56	8000

Table 3.1: Data for different leg muscles, taken from a cycling program from de VU Amsterdam.

Optimal force F_o (N): the maximal isometric force. It is calculated as the product of the PCSA and *specific tension*[7]. The *specific tension* for humans is about $31.8\frac{N}{cm^2}$.

Optimal tendon length $L_{o,T}$: the tendon length when optimal muscle force acts on it. Tendon has a very adaptive behavior. When its resting length $L_{0,T}$ is given we can approximate it with the optimal length $1.04L_{0,T}$.

Maximal muscle-tendon length $L_{max,mtc}$(cm): the maximal length of the whole mtc.

Activation threshold U_r: fractional activation level threshold. If the input activation U exceeds this level all fascicles or units will be recruited and fire action potentials. At $U = 1$ all fascicles will fire at the maximum fire frequency.

Maximal fascicle length $L_{max}(L_0)$: the maximal fascicle length is calculated with $\frac{L_{max,mtc}-L_o}{L_o}$.

Gathering the muscle data is one of the most difficult steps we encountered. The Yamaguchi table [12] is the most well used table. If we want to incorporate the muscle models with the skeleton model, we should also know their length properties as function of the skeletons configuration. In brief, the skeleton model and the mtc's should fit together. This table however does not contain those needed data. Therefore we used muscle data from a cycle program [Soest]. The data for different mtc's of the leg are listed in table 3.1. As we see we are missing some parameters. Most of the missing parameters can be derived however. The only parameters that we do not have are the U_r and L_{max}. We choose $U_r = 0.8$ for all mtc-s and we guessed $L_{max,mtc}$ for each mtc such that $L_{max} \approx 1.3$.

Building units

Real skeletal muscles consist of many units. They can easily sum up to more than 100. Each unit behaves differently within the muscles. The way they are recruited depends on the size and fiber-type of their fibers. The complexity of the model scales with the number of units we want to model. Remember that we need 4 state variables for a unit. Is is very common to group similar units into big units. BuildMuscles can automatically divide the whole muscles into units.

[7] In cartoons we associate thick muscles with lots of power.

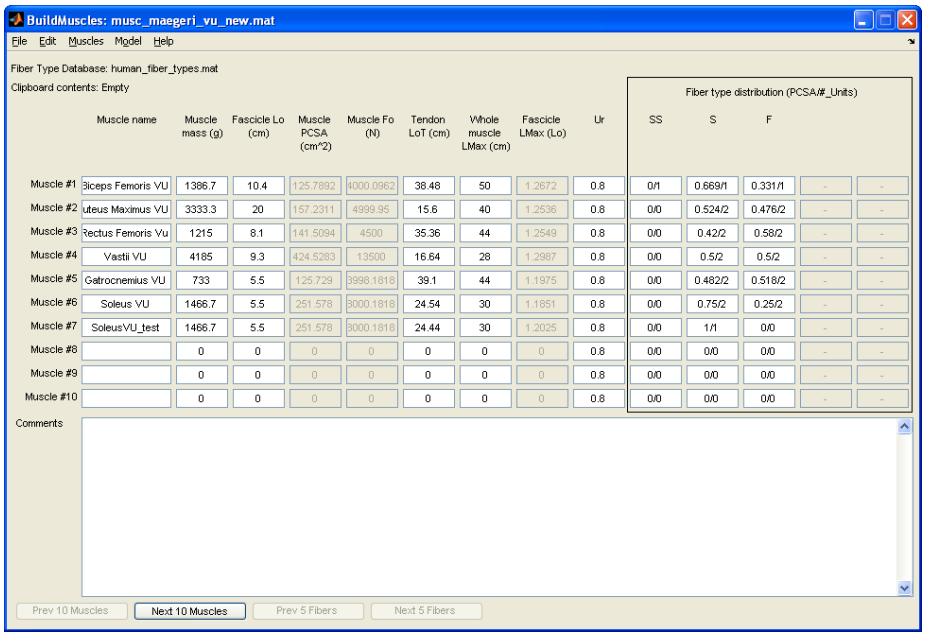


Figure 3.8: With the program BuildMuscles we can build and export mtc-s as Simulink block after we have filled in necessary data. Furthermore it can divide the whole muscle into different units.

We should fill in the fiber distribution is the whole muscle and the number of units we want. The higher the number of units the more computation intensive the simulation will be. The fiber-type distribution are taken from the Yamaguchi [12] table. Once the number of units are filled in we can export the mtc-models as Simulink blocks. In these blocks the muscle equations are implemented. In A we see how these equations are implemented. The equations are given in [10] and we verified the Simulink block with them. During our verification we discovered a small error in the Simulink blocks. Two force signals were switched(see figure 3.9). Although it was very easy to fix it manually, we fixed the source-code of BuildMuscles. The reason is that we need to build more mtc models and fixing them one by one costs time.

3.7 Coupling the skeleton and mtc-models

The skeleton model outputs joint-coordinates and needs net-joint moments as input, whilst the mtc-models outputs muscle force and needs mtc-lengths as input. If we want to couple both models we have to calculate the mtc-length and the moment-arm of the mtc's force [8]. The mtc-lengths and moment-arm both can be calculated with Grieve method(see appendix B). In figure 3.11 we see our previous thigh model coupled with the rectus femoris model. The rectus femoris is a bi-articular mtc(see figure 3.10). [9] It flexes the thigh and extends

[8] moment = a*F

[9] In mechanical systems torque-drivers are used to activate the joints. They only cover one joint.

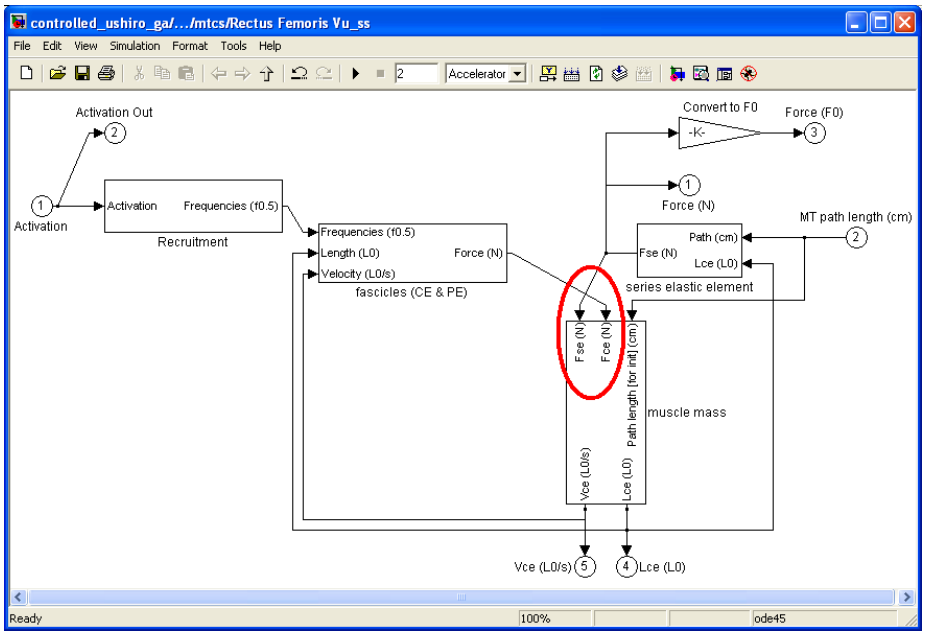


Figure 3.9: A small error that comes with Virtual Muscle 3.1.5. In this picture the error is already fixed. In the original blocks the force signals were interchanged. These forces are used to calculate the length of the CE elements.

Figure 3.10: A lateral(side) view of rectus femoris. The rectus femoris flexes the hip and extends the knee. Source:http://www.fotosearch.com.

Name	Rectus Femoris
Mass(kg)	1.215
F_{max}(N)	4500
$l_{oi,max}$ (cm)	44
$l_{ce,opt}$ (cm)	8.1
$l_{se,opt}$ cm	35.36
Grieve const.(hip) (A_0, A_1, A_2)	(0.110, 0.035, 0)
Grieve const.(knee) (A_0, A_1, A_2)	(0.308, 0.042, 0)
nr. units	1

Table 3.2: The data we used for our rectus femoris. The grieve constants are needed to couple the rectus femoris with the thigh. We only use one unit to model the whole muscle group.

the knee. Because we do not have a knee we assume the knee to be at zero degrees for the rectus femoris. At this position the virtual shank is thus aligned with the thigh. The rectus femoris flexes the hip. If it is activated and the generated moment is larger than the gravitational moment the thigh will rise. In table 3.2 we see the data we used to model and couple the rectus femoris.

Therefore we named the model raising_thigh.

3.8 Experiments

Now that we have coupled the skeletal model with a mtc-model we can start experimenting. In all the experiments the initial angle is set to zero, that is trunk and thigh aligned. The solver is set to ODE45 with $reltol = 10^-6$

Experiment 1: varying the activation.

In these experiments we use a constant activation $act(t) = c$ as input and simulate the model for 2 seconds. In figure 3.12 we see the result for $c = 0.0, 0.8$ and 0.9.

Experiment 2: varying mass and inertia

One of the greatest advantages of musculoskeletal models is the ability to do practically impossible experiments. This time we vary the mass and thus inertia. The activation function is set to $u(t) = 1$. In figure 3.13 we see the results for the normal thigh, a thigh with twice the mass and a thigh with halve the mass.

Experiment 3: different activation functions

In reality the activation function is very complex with varying amplitudes and frequencies. In this experiment we used some standard functions.

Constant $u(t) = 1$

Linear $u(t) = 0.8 + \frac{.2}{2}t$

Sinusoid $u(t) = 0.9 + 0.1\sin(\pi t)$

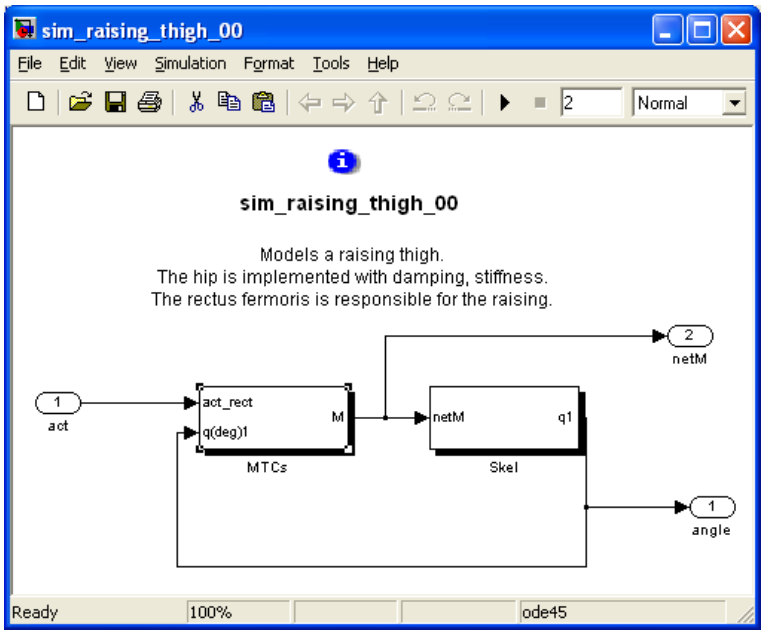


Figure 3.11: The raising_thigh model. Many blocks are masked to get a clear view.

The results are given in figure 3.14.

Conclusion

After all these experiments we are wondering if we can control the thigh. Can we keep it still for example. This can be seen as a control problem, but also as an optimization problem. We want to minimize the difference between the output angle and a reference angle. In the next chapter will will use a self-implemented genetic algorithm to find an activation function that can do this.

3.9 Project Mae-Geri

It was very tempting to extend the thigh-model. Therefore we built a leg model that can simulate the mae-geri or front-kick. It is a 3-segmented skeleton model with 6 mtc models. The most difficult part was to couple the skeleton model and the mtc models. Once this was done we can start simulation. A major problem is that we did not know what activation we should feed the model, besides zero and maximal activation. In figure 3.15, 3.16 and 3.17 we see some details of our leg model. The most difficult part was to couple the skeleton and the mtcs. During simulation with maximal activation for each mtc, we encountered integration errors. The precise reason is unknown, but we suspect a bad initial state to be the cause of this. In the initial state the system should be in rest. Because there are 6 mtcs and 3 bodies, finding a proper initial state is not easy. In this thesis we did not solve this problem.

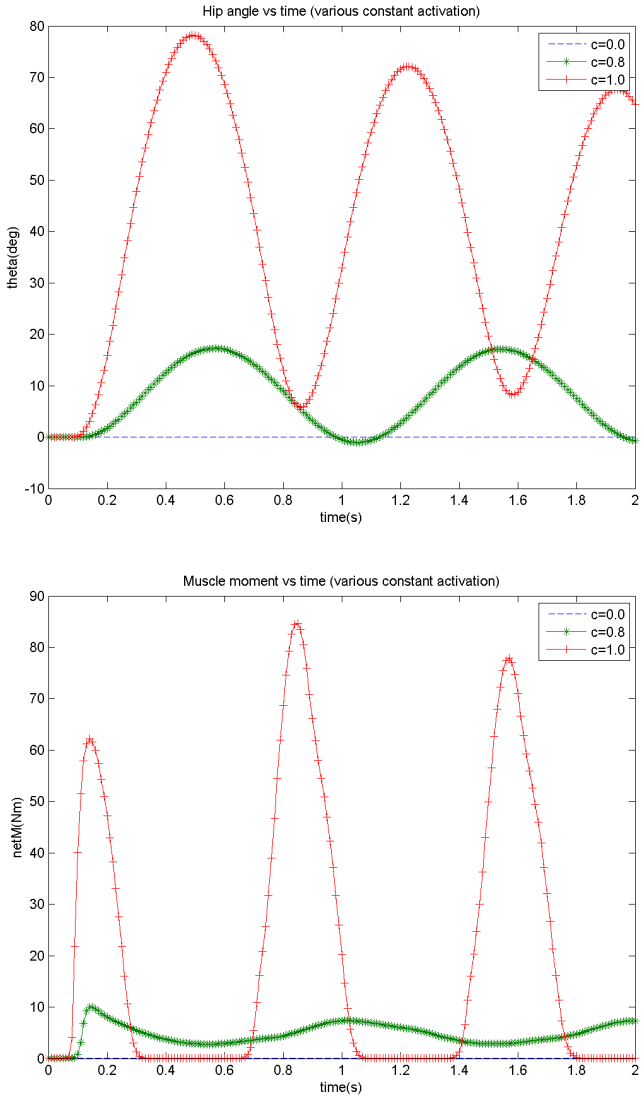


Figure 3.12: We used different constant activation functions to activate the rectus femoris. Top: the hip angle. Bottom: the generated moment. The moment-arm is a constant(see table and Grieve constant). Therefore the moment graph can be seen as a scaled force-graph. When the activation is 0, the thigh will not move and the angle is thus zero all the time. Our mtc models consist of 1 fascicle with fast twitch fibers. The activation threshold is set to $u_r = 0.8$. At $act = 0.8$ the fascicle then becomes activated and starts to fire action potentials at its lowest frequency. With $act = 1.0$ the stimulation is maximal and the fascicle will fire action potentials at its highest frequency. With $act = 0.8$ and above we expect the thigh to raise. That it will fall afterwards and raise again and so on we honestly did not expected. We might explain this with the force-length and force-velocity relations of the fascicle. During the raising period the length will decrease and the contraction velocity will increase. All this will result in less force and gravity will take over. When falling however, the fascicle length will increase and we also have eccentric contraction.

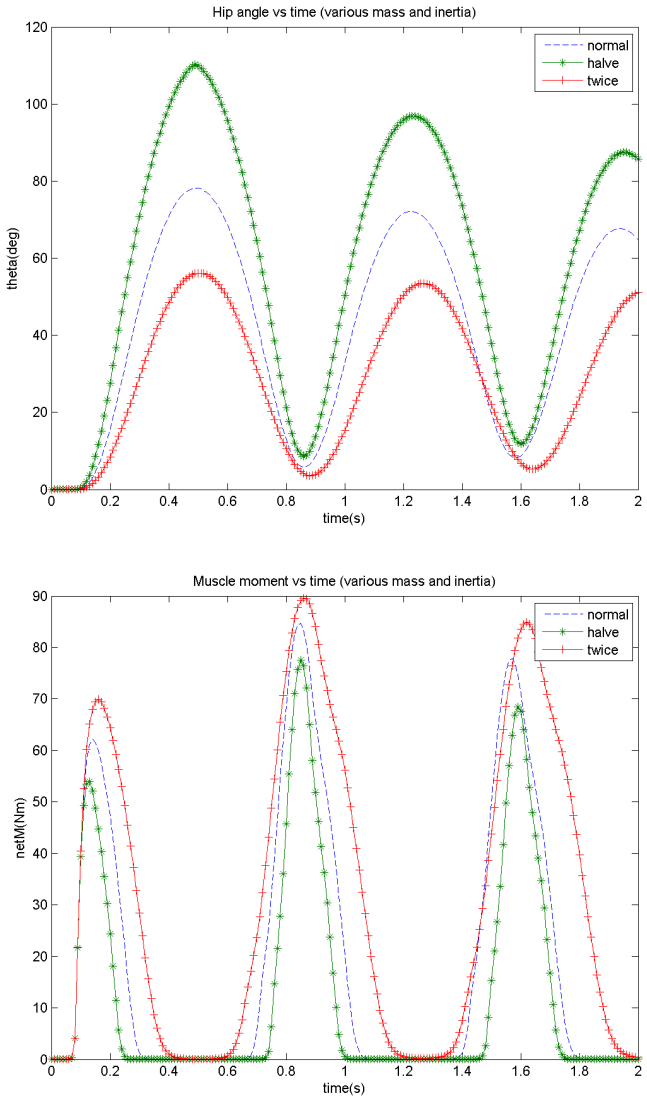
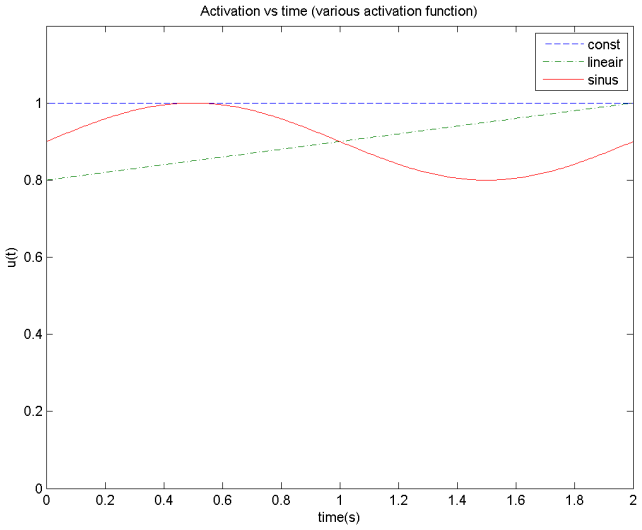
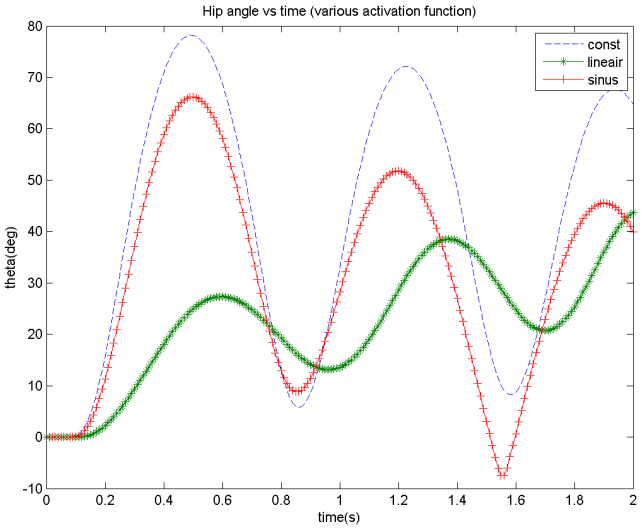


Figure 3.13: We changed the mass and thus the inertia of the thigh. The lighter the thigh, the higher it will raise. This agrees with our expectations.

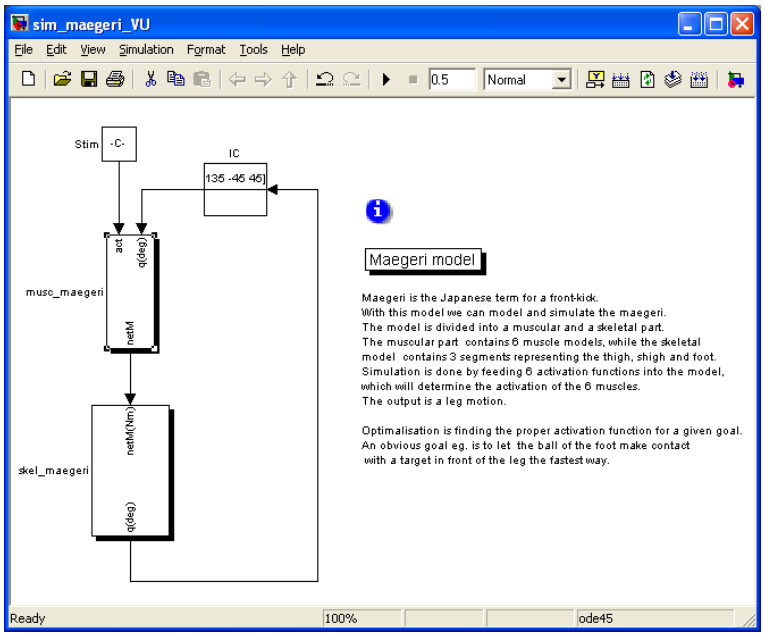


(a)

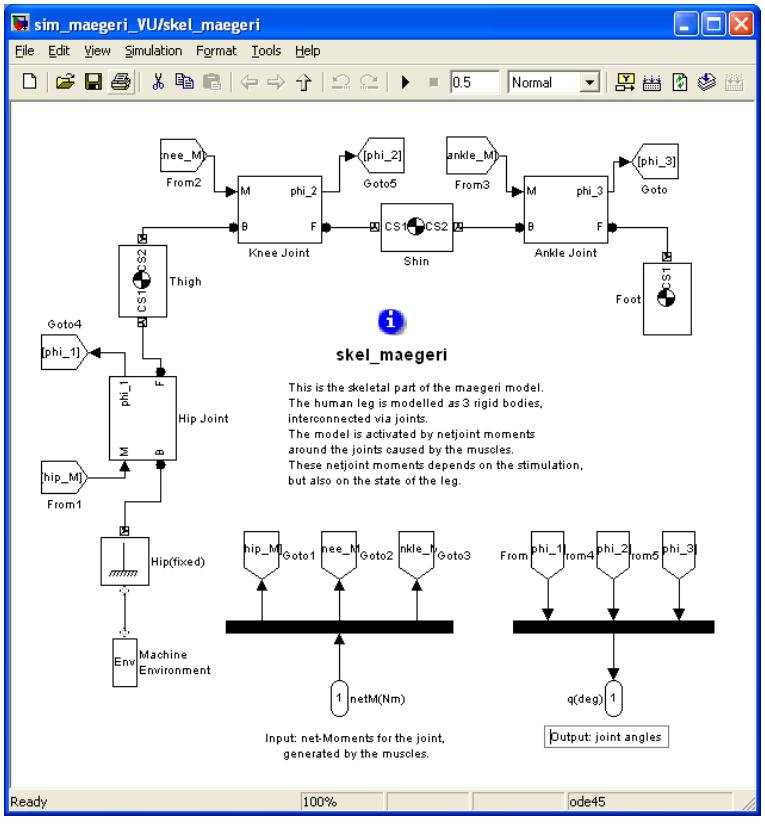


(b)

Figure 3.14: Different types of activation functions. (a) The activation functions. (b) The joint angle θ. We can see some relations between activation and joint angle.



(a)



(b)

Figure 3.15: The top level view of our leg model. (b) The skeletal sub-system, which is an extended version of the thigh-skeletal model.

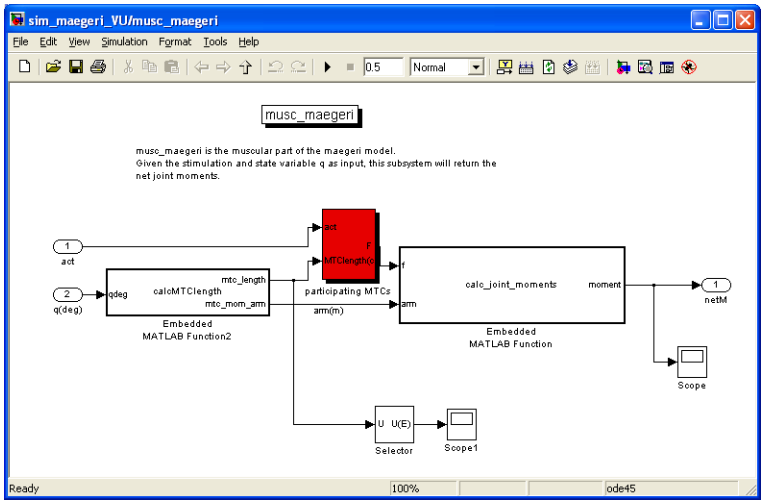


Figure 3.16: The musculo-system of the leg model. The mtc-lengths and the moment-arms should be calculated to allow interactions with the skeletal model.

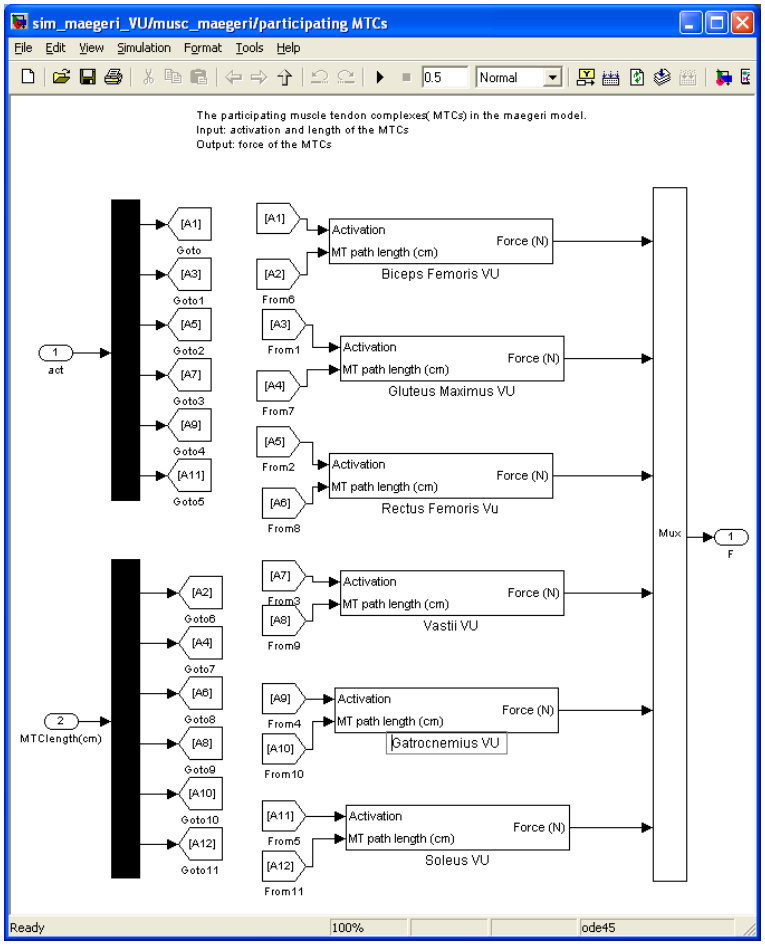


Figure 3.17: The participating mtcs for our leg model. There are 6 of them.

3.10 Conclusion

We have seen how we can build models for skeletal muscles or muscle-tendon complexes(mtcs) with Virtual Muscle. The force-producing behavior is very complex and currently there is no theory that describes the behavior. Virtual Muscle models the mtcs empirically. We hope that the behavior can be extrapolated and thus get realistic results for our musculoskeletal system.
A very hard problem is to get realistic data for the mtcs. If we want to build musculoskeletal models, the skeletal model and the mtcs should fit together. To obtain these data lots of practical measuring experiments should be done. For us this is almost impossible. Luckily we have data from a jumping program supplied by the VU Amsterdam.
Virtual Muscle exports its mtc models as Simulink blocks and we can couple them relatively easily with skeletal models that we build with SimMechanics the Simulink environment. The simulation is then very easy. We have added a hip flexor to our thigh model from the previous chapter and done some experiments with it.
The outputs of our thigh model were very unexpected, except for the case where we use $act = 0$ as input. Furthermore we do not know what activation function we should feed the model. It is amazing how the human mind can do all kind of amazing motions with these complex force producers. We should however not forget that we needed years of training to do "simple" movements like walking for example.
From a mathematical point of view the behavior of our musculoskeletal system is very nonlinear. Optimizing the output is therefore a real challenge. In the next chapter we will see how we can use evolutionary algorithms to perform optimizations.

Chapter 4

Optimizations with Evolutionary Algorithms

4.1 Optimizing our motions.

We have built a musculoskeletal model and can perform all kinds of experiments with it. The input is an activation function *act* for each mtc-model. The output vector $\vec{Y}$ contains variables we are interested in. Many times we want to find an optimal output. Examples: fastest kick, economic kick etc. To do this we should first define an objective function $\mathcal{F}$ for $\vec{Y}$. It is obvious that depending on one's interest many objective functions can be defined. Then we introduce a function h that maps *act* to $\mathcal{F}(\vec{Y})$. h is a very complex function and with each simulation we can only know one point of h. Our optimization problems become as follow: find *act* such that h is optimal. Figure 4.1 makes it all more clear. In this chapter we will use Evolutionary Algorithms to find the optimal *act*.

4.2 Evolutionary Algorithms(EAs)

Evolutionary Algorithms look at a number of possible solutions for a given problem. The collection of these solutions is called a *population*. The population

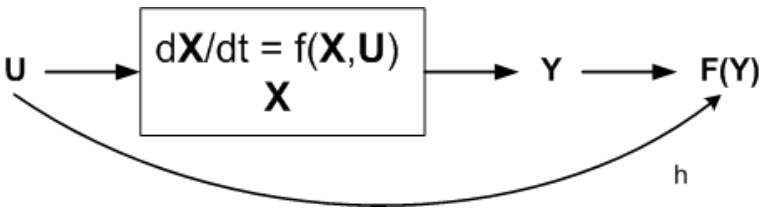


Figure 4.1: Schematic overview of optimizing our system. For a given input $\vec{U}$ (*act* in our case) we get an output $\vec{Y}$ which we will rate with $\mathcal{F}$. Our optimization problems can be seen as finding *act* such that $h(act)$ is optimal.

will evolve over time until the problem is solved or the maximum processing time is met. So at each time step we have a different generation, supposedly to fit better to the problem than its predecessors. An evolution step is done by applying several evolutionary operators on it. These operators mimic the behavior of nature's evolution processes. The main operators are:

Selection: Select the better solutions from the current generation. Whether a given solution is better than another is determined by a user input function. The objective function.

Recombination: Solutions are created by recombining the selected members from previous generations.

Mutation: Complex problems tend to have many local optima. With mutation we want to escape from a local optimum. Mutation is done be changing some solutions slightly.

These operators make that evolutionary algorithms differ from a simple random search. Good information are passed through next generations. For each operator we have different choices. Furthermore they are parameterized. In appendix C we see an overview of the operators that we have implemented. These operators are implemented from [14].

4.3 Terminology

We assume that our audience are familiar with EA. Below are some terminology we used.

Population: A set of potential solutions for our problem. With EA we want to evolve the population to a better one and so one, until termination. Mostly we are only interested in the best solution when the search is terminated. Sometimes we are also interested in other good solutions(local optima). The size of the population is denoted with N.

Region: Several publications have shown that it is better to split the entire population into a number of sub-populations or *regions*. More in **??**

Solution: A solution is an array of *variables* that represents the real solution. In biological terms we can compare a solution with a *chromosome*. [1]

Variable: A solution is an array of variables. When we used the term *chromosome* for a solution, then the term *gene* is the equivalent for variable. The three most common types of variables are: *binary-valued*, *multi-valued* and *real-valued*.

Objective function: a function that measures how good a solution is.

[1] EA and GA are inspired by processes in nature. Some EA specialists nowadays want to break the link with biological terms.

4.4 The basic Evolutionary Algorithm

The basic EA algorithm is listed algorithm 3 and is quite powerful to solve various optimization problems. EA has some major drawbacks however. First we do not have certainty that the search will converge to an optimal solution. This because it is a stochastic search method. Second EA is very time consuming. We easily speak in terms of thousands of fitness evaluations. In our case one fitness evaluation means one simulation. Luckily they are very well suited for trivial parallel implementations, where we can use multiple computer processors simultaneously. Other methods and operators can be used to extend the basic algorithm and improve its performance.

```
Input: Fitness function f
Output: Best solution found
population =Initialize(f, random) ;
while terminal constraint is not met do
    parents = select(population, f) ;
    children = recombine(parents) ;
    children = mutate(children) ;
    population = reinsert(parents, children) ;
end
return(Best solution(population))
```

Algorithm 3: The basic Evolutionary Algorithm(EA).

4.5 Extending the basic EA

The basic evolutionary algorithm is able to solve many complex problems. Their performance can be improved by good choices for the GA-operators and their parameters. When the fitness functions are cheap we can afford creating large populations and allowing large evolution time. Sometimes we have very expensive fitness functions and we have to use more sophisticated EA. A simulation is then very costly and with better EA we hope to get better results with less computation. Below are listed 2 extensions to the basic EA.

4.5.1 Multiple regions

With this method we divide the population into several sub-populations. Each of them will evolve independently for some generations. The number of generations where the sub-populations evolve independently is called *isolation time*. After each isolated evolution the subpopulation will exchange their best solution with each other. This is called *migration* and there are several ways to do this. After migration the sub-populations will evolve independently and so on. In algorithm 4 we see the algorithm for EA with different regions.

```
Input: Fitness function f
Output: Best solution found
regions = Initialize(f, random) ;
while terminal constraint is not met do
    foreach region do
        region = evolve(region,n,g);
    end
    regions = migrate(regions);
end
return(Best solution(population))
```

Algorithm 4: The different regions EA, which is an extension of the basic EA.

4.5.2 Different strategies.

The EA with multiple regions can be seen as an extension of the basic EA. It can be extended further. In 4.5 we have mentioned that we can improve the basic EA by choosing the right operators and their parameters. A given set of choices for EA operators and their parameters is called an evolution strategy. In most cases we do not know which are the better ones. One solution to solve this problem is to give each subpopulation a different evolution strategy. By doing this we bet on more horses and thus hopefully have more chances for winning. Furthermore we believe that with different strategies the subpopulation will help each other. The most obvious example is the case where we use different mutation strategies. In the beginning the strategy with large mutation will do better, while after some time and migrations the strategy with small mutations will do better. The algorithm for EA with different strategies is given in algorithm 5.

```
Input: Fitness function f, various evolution strategies: [EA strategy]
Output: Best solution found
regions = Initialize(f, random) ;
while terminal constraint is not met do
    foreach regions do
        region = evolve(region, EA strategy);
    end
    regions = migrate(regions);
end
return(Best solution(population))
```

Algorithm 5: The different strategies EA, which is an extension of the differen regions EA.

4.6 Project Static Thigh

In the end of the previous chapter we wondered if we could keep the thigh still. The reason for this is to find a so called *operational point.* In this point the system should be in rest. We can save the state of this point and start other experiments from this point. In project Static Thigh we want to raise the thigh from an initial state and keep it still at 10 degrees for a short period of time. The initial state was chosen to be the state where the thigh is at rest $\dot{\theta} = 0$ and perpendicular to the ground($\theta = 0$). This can be seen as an optimization problem where we want to minimize the difference of the hip angle θ with a reference angle θ_{ref} for a certain amount of time. Can the EA solve it?

4.6.1 Using EAs

If we want to use EAs to solve this problems we should do the following:

1. Formulate the objective function $\mathcal{F}$: this function tells us how good the simulation is. It is defined in terms of the model's output.
2. Encode the solution: our solution is an activation function *act*. This function should be encoded into an array of variables. The evolutionary operators depend on what type of variables we use.
3. Choose Evolutionary Strategy(s): if we use the basic EA we only have to choose one evolutionary strategy. If we use the different strategies EA we should of course choose more evolutionary strategies.
4. Perform evolutionary search: we set some stopping criteria(goal reached, maximal search time etc.) and start the search.

Formulating the objective function

The objective function tells us how good a solution is. The definition of objective functions is problem dependent. The EA only gives solutions for what we asked for and therefore a well defined fitness function is of paramount importance. This time we want then the thigh to move up and stand stationary from $t = 0.75s$ to $t = 1.0s$. We choose the objective function $\mathcal{F}$ to be the *r2-norm* of $(\theta(t) - 10)$ for $0.75 \leq t \leq 1$. We have:

$$\mathcal{F} = \sqrt{\int_{t=0.75}^{1} (\theta(t) - 10)^2 dt} \tag{4.1}$$

We can only approximate the objective function. We let the model produce output at fixed time-points[2], that are equally distributed along the time-axis with distance dt. The objective function is then approximated with:

$$\mathcal{F} \approx \sqrt{dt * \sum_{i=0}^{N-1} (\theta(t_i) - 10)^2} \tag{4.2}$$

with $t_0 = 0.75$ and $t_N = 1$.

[2]With variable time-step solvers linear interpolation is used to produce results at specified time-points. This will result in extra approximation errors!

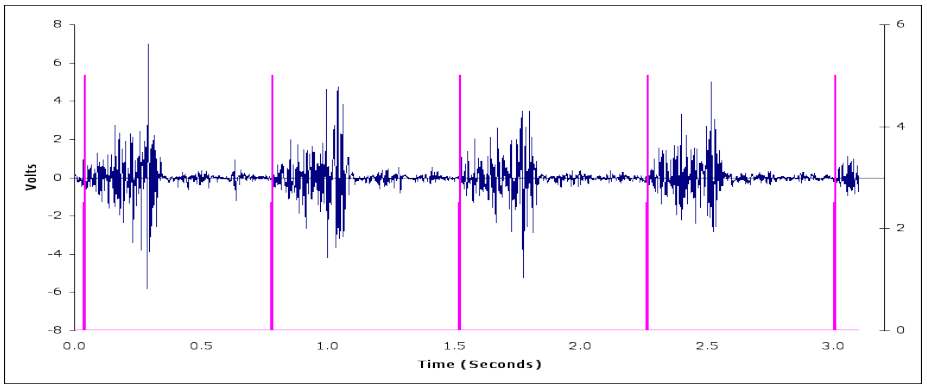


Figure 4.2: A typical EMG signal, Voltage vs. time. The electrical activity in the muscles. The amplitude tells us how active the muscle is and can be related to the neural activity. If we want to use this signal we should clip the signal to [0,1].

Encoding the activation function

Although the objective function is very important, formulating it was fairly easy. The next step is encode the solution, which is the activation function. This is a difficult step, because the activation function is a very complex function. It stands for the normalized neural activation function. If we want to do it right we should model the neural activation and parameterize this function. A solution can then be seen as an array of the parameters that determine the activation function. According to the manual of Virtual Muscle 3.1.5 [11], we can use recorded electromyography(EMG) as activation. EMG measures the sum of all synchronous currents in each muscle fiber. In figure 4.2 we see a typical recorded EMG signal. Currently our knowledge of the neural activation function is too low and therefore we will use a simplified function to model the activation function. We have chosen our activation function to be made up with piecewise constant functions, that we call *pulses*. A pulse can only be zero or one. All pulses have the same width and that depends on the number of pulses that we use. The *frequency* f is a quantity for the number of pulses and is defined as the number of pulses per second. The width w of a pulse is then $w = \frac{1}{f}$. Now a solution is an array of binary valued variables. Because we want to simulate the thigh for one second, the size of the array equals f. In figure 4.3 we see the case for $f = 4$. Suppose we have an array of pulse of size f. Then we can calculate act with:

$$act(t) = \begin{cases} \sum_{i=0}^{f-1} p(i)\alpha_i(t) & \text{if} \quad 0 \leq t < 1 \\ p(f-1) & \text{if} \quad t = 1 \end{cases} \tag{4.3}$$

where

$$\alpha_i(t) = \begin{cases} 1 & iw \leq t < (i+1)w \\ 0 & \text{elsewhere} \end{cases}$$

and $p(i)$ is the i-th variable of the pulse-array..

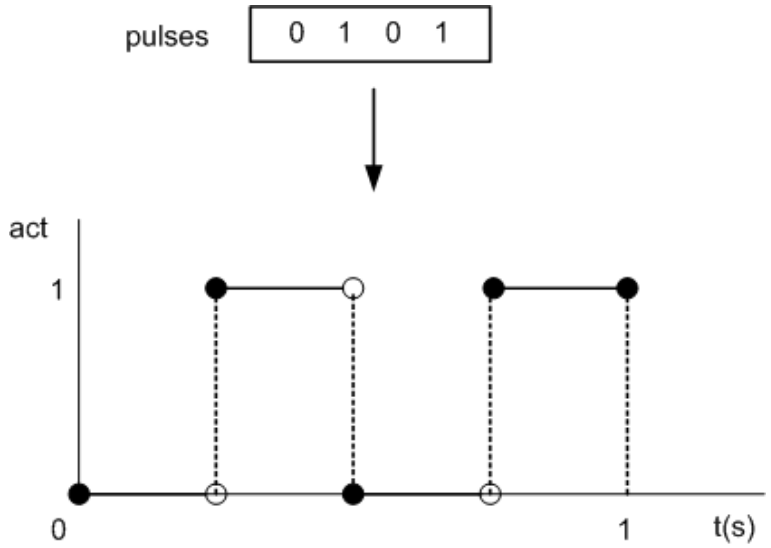


Figure 4.3: A binary valued pulse function with frequency $f = 4$. $T_{max} = 1$ will get the value of the last pulse.

The extended raising-thigh model

The EA will find solutions in the form of pulse arrays. These pulse arrays should be converted into activation functions. In the extended raising-thigh model we let the thigh-model do the pulses to act conversion. Suppose we have solution p which is an array of length f, then the activation function can be calculated with equation 4.3. By doing this we guarantee ourselves that the activation function will not be interpolated. In figure 4.4 we see a screen-shot of the extended model that we named sim_act.

Benchmarks

With the choice of EAs to solve the problem, a large number of simulations should be done. Usually we speak in terms of 10 thousands. Therefore we did a little benchmark with our computer(AMD 2600+, WindowsXP) for different activation, solvers and relative tolerances. The results are given in table 4.1 and 4.2. We are not happy with these results. Each simulation takes around 1.8 seconds. Because there are no very big time differences, we used the ODE45 solver with RelTol $1e^{-6}$ for our simulations. In this way we hope to get simulations with a relatively high accuracy.

4.7 The experiments

We let the model produce θ at specified time-points. The distance dt between these time-points is fixed and equals $1/1000 = 10^{-3}$. The used solver is the ODE45 variable time-step solver with relative tolerance 10^{-6} which is quite accurate. The approximation of the objective is calculated with equation 4.2. Output θ and thus interpolated values of θ are used. This will result in extra

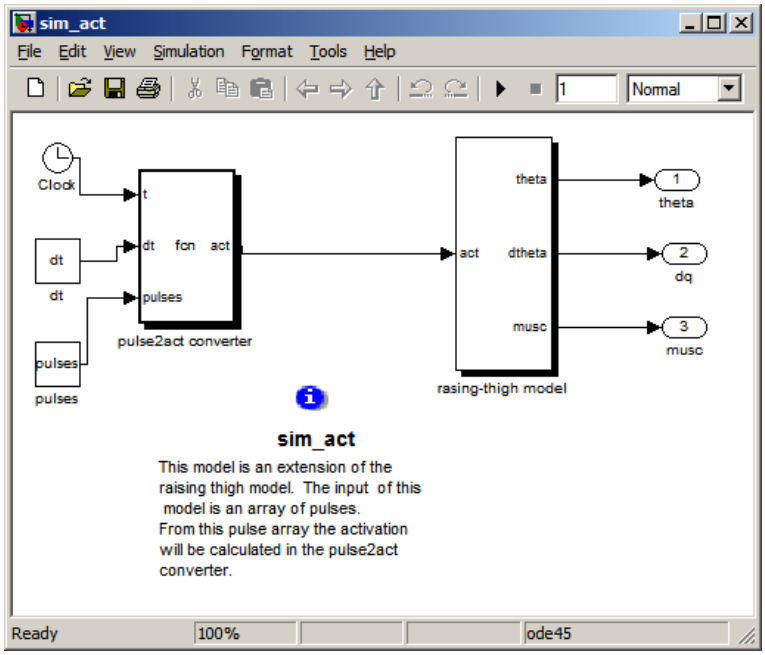


Figure 4.4: The extended thigh-model that we named sim_act. The input are the found pulses('solutions') and the pulse2act block will calculate the activation function at needed time-points.

	With ODE23 time(s)		
RelTol	pulses =1	pulses = 0	pulses = rand
10^{-3}	2.8224	1.7862	1.6384
10^{-4}	1.7267	1.8196	1.6757
10^{-5}	1.7357	1.7251	1.7108
10^{-6}	1.7580	1.7340	1.7459

Table 4.1: The time it takes for our computer to evaluate a fitness function. The frequency is set to $f = 40$. The pulse array is thus an array of length 40. pulses = 1 means all variables are 1, pulses = 0 means all variable are 0, pulses = rand means the variables are randomly chosen to be 0 or 1. Once the array of random variables are generated it is used over and over again for the benchmarks. In this table the ODE23 solver is used. The time is the average of 10 simulations. There is not much time difference between different $relTol$. When we used our EA, we should count on around 1.8 seconds for an objective-function evaluation. This is a serious bottleneck.

	With ODE45 time(s)		
$relTol$	pulses =1	pulses = 0	pulses = rand
10^{-3}	1.8420	1.3410	1.7662
10^{-4}	1.8205	1.4211	1.7656
10^{-5}	1.8680	1.3953	1.8378
10^{-6}	1.8874	1.3657	1.9026

Table 4.2: Same as 4.1, but with ode45. In most cases a smaller $relTol$ needs more time. But there are no big differences. Comparing with the ODE32 table we see that sometimes ODE23 needs less time, but sometimes more. We are more interested in pulses of type rand. There we see that ODE45 needs more time, about 5%.

approximation errors and we should keep this in mind. We directly use the different strategies EA, because we think that these will do better.

4.7.1 Experiment 1: $f = 40$

We used a pulse array with frequency $f = 40$. The values of the pulses are 0 or 1. Below are the basic settings for the EA.

- number of regions: 4
- region-size: 40 solutions
- migration-time: 5 generations
- a unique EA-strategy for each region.

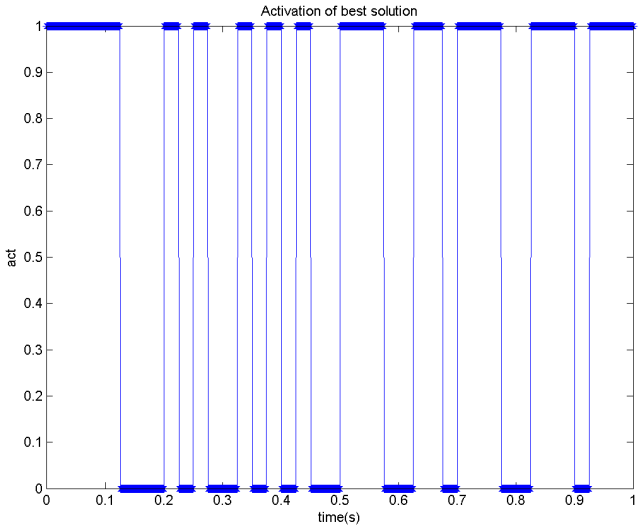
The search is stopped if the the best solution did not improve after 3 migrations. Results of the search are saved and the search restarted with random solutions as initial solutions and so on. We let the search run for one night and in figures 4.5 and 4.6 we see the best result after 9 restarts.

The best found objective is 0.0224 which is quite small. Wether this is a local or global optimum we do not know. We can know it of course if we evaluate all possible solutions that equals 2^{40}. If we assume that it takes around 1.8 seconds for 1 evaluation for our computer, then we should wait around 6×10^4 years. If we use lots of computers however we can have the answer within our lifetime. Before we try to do this or think of trying to do this, we will do another experiment first.

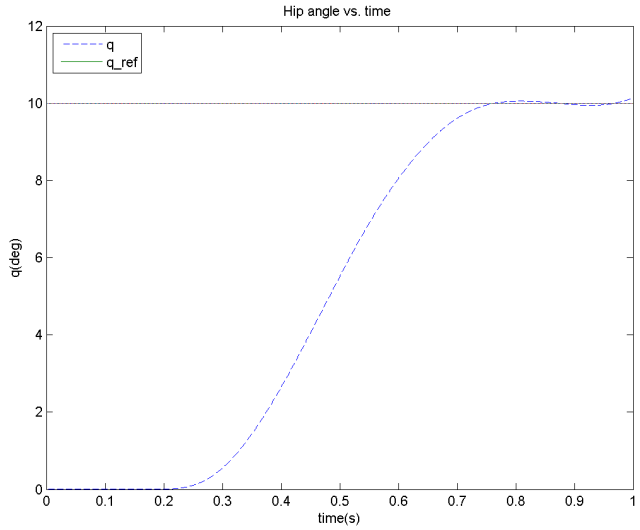
4.7.2 Experiment 2: $f = 80$

This experiment is the same as the previous one, except we changed the frequency to $f = 80$. With $f = 80$ the search space is much larger than with $f = 40$(around 10^{12} times) and maybe we can find a better solution. We also let our computer do the search for one night and the result after 4 restarts(around 8.5 hours) is given in figures 4.7 and 4.8. The number of restarts is lower than the case where $f = 40$. This can be explained with the increased search space if we take $f = 80$.

The found objective is smaller than with $f = 40$ (0.0180 vs. 0.0224) and the plot of the found θ vs. time looks better. If we look at the found activation

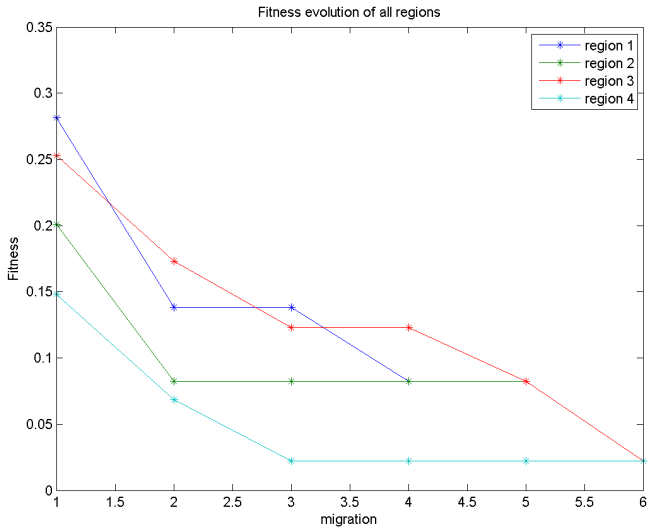


(a)

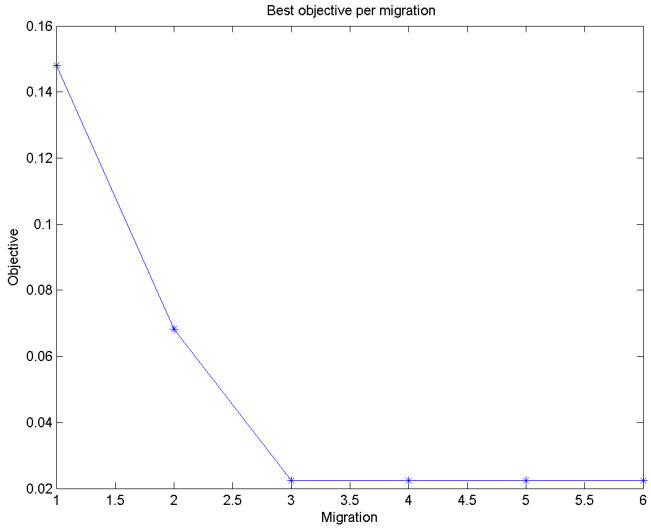


(b)

Figure 4.5: Best result of experiment 1. (a) The found solution(activation function). We can not see a clear pattern in it. (b) The found θ vs. time. It is close to what we want and it look quite good if we also take numerical errors in mind.

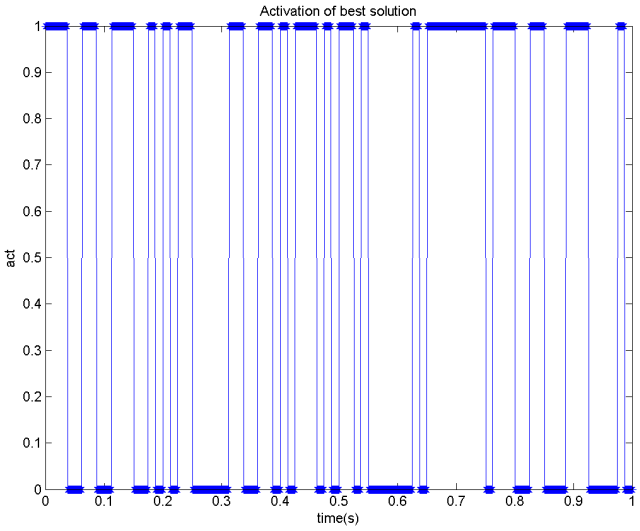


(a)

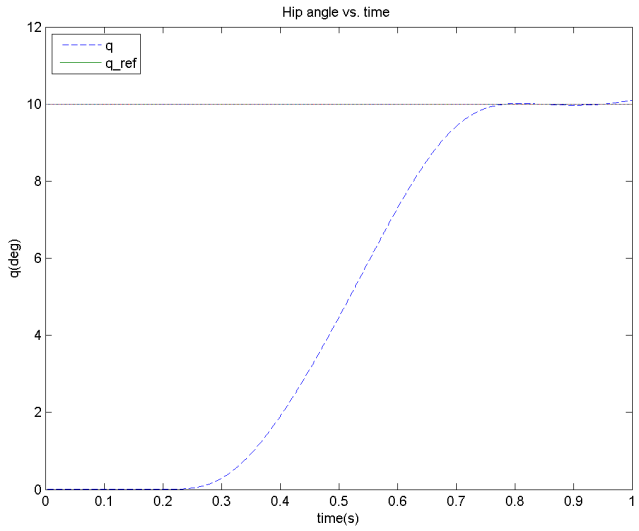


(b)

Figure 4.6: Best result of experiment 1(continued). (a) The evolution of the best solutions of each regions. Here the approximated objective vs. migration is plotted. The migration method we used is the universal migration method. With this method each region will choose another region at random and replace the worst solution of the chosen region with its own best solution during migration. (b) The evolution of best objective of the whole population(all regions). Because of elitism(best solutions are saved for next generations) we are guaranteed of a monotonous decrease for the objective. The best found objective is around 0.0224

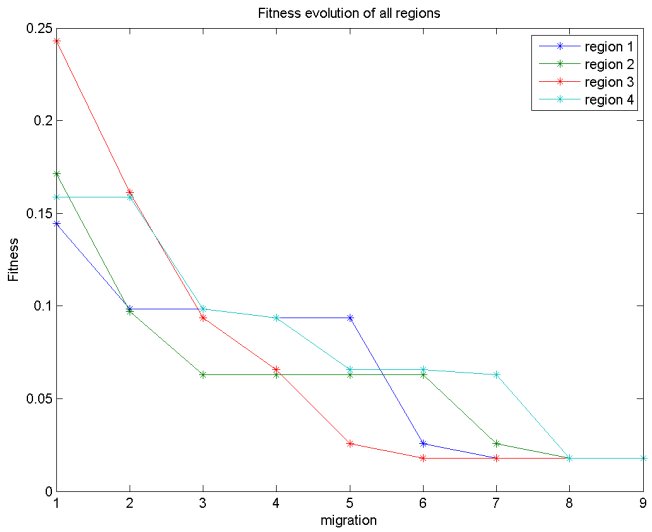


(a)

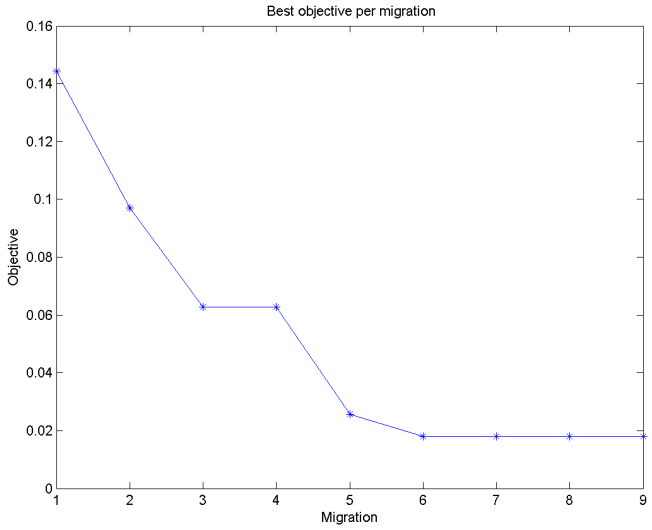


(b)

Figure 4.7: Best result of experiment 2. (a) The found activation function, which is a bit mysterious. (b) The θ vs. time belonging to the best solution. It looks good.



(a)



(b)

Figure 4.8: Best result of experiment 2 (continued). (a) The best-objective evolutions of the 4 different regions during the best search. The time is given in migrations. A migration counts for 5 generations. (b) Evolution of the best objective of the whole populations. The best found objective = 0.0180.

$relTol$	$step_{min}$	$step_{max}$	t_0	$\mathcal{F}_2$
$1e^{-2}$	$4e^{-20}$	0.014	0.75	0.0196
$1e^{-3}$	$3e^{-19}$	0.012	0.75	0.0180
$1e^{-6}$	$7e^{-18}$	0.008	0.75	0.0176
$1e^{-9}$	$4e^{-18}$	0.008	0.75	0.0175

Table 4.3: The results of simulating with the best found solution for different relative tolerances $relTol$. $step_{min}$ and $step_{max}$ is the minimal and the maximal step-size respectively that the model used. t_0 is the start time that is used to calculate the objective. Surprisingly $t_0 = 0.75$ for all $relTol$. This indicates that very small time-steps are used around $t = 0.75$. $\mathcal{F}_2$ is the new objective approximation where we also take the variable time-steps into account.

functions of both solutions we can not really tell the difference. An interesting way to compare both solutions is by looking at the modelled tendon-force that belongs to these solutions. In figure 4.9 we see the plots.

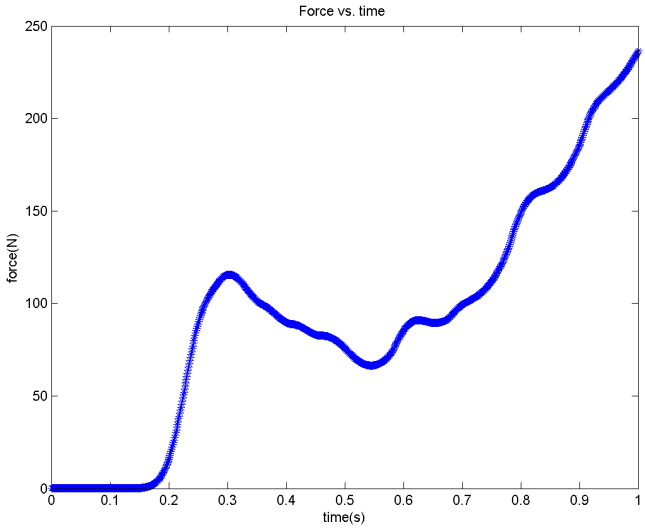
4.7.3 How reliable is our best solution?

With the 2 experiments we have found 2 solutions. Based on the objective of these solutions we declare the found solution of experiment $2(f = 80)$ as winner. Because we used numerical results there will always be numerical errors(rounding off, numerical integration, interpolation etc) involved. Can we trust our results? The simulations in the EA searches used the ODE45 variable time-step solver with relative tolerance $relTol = 10^{-6}$. The reason for choosing a variable time-step solver is efficiency. The solver will only use smaller time-steps when needed. For the calculation of the objective function however we used interpolated θ. This will lead to numerical errors, and the question is how bad it is. Therefore we examined the best found solution thoroughly. We feed this solution to our model and simulate the motion a few times. Each time the ODE45 solver is used, but with different relative tolerances. The calculated θ instead of the interpolated θ is then studied. The objective function is now approximated with:

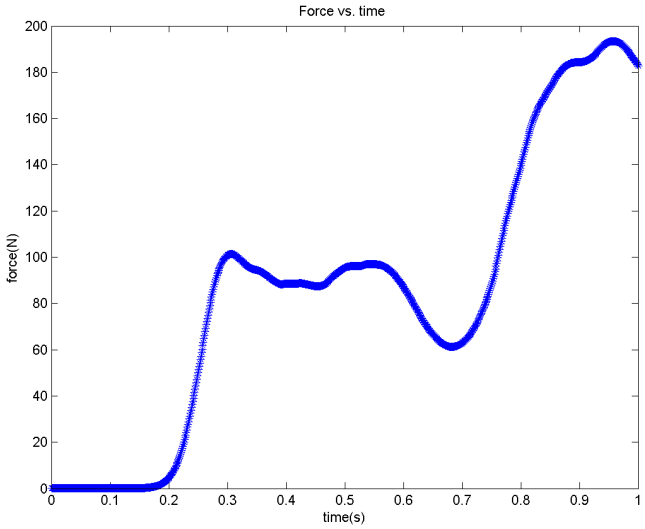
$$\mathcal{F}_2 \approx \sqrt{\sum_{i=0}^{N-1} dt_i * (\theta(t_i) - 10)^2}, \qquad \text{with} \qquad dt_i = t_{i+1} - t_i \tag{4.4}$$

where $t_0 \approx 0.75$ and $t_N = 1$. Relevant quantities of these simulations are given in table 4.3. In this table we see that $\mathcal{F}_2 = 0.0176$ ($relTol = 1e^{-6}$) and does not differ much from $\mathcal{F} = 0.0180$ which is good news. It means that the interpolation errors are quite limited[3]. In figure 4.10 we see a plot of the calculated θ vs. time, where we zoomed in at $t = 1$. There we see that solving with a $relTol = 1e^{-6}$ is accurate enough. Then there is the question: is the activation function really used? The width of our pulses equals $\frac{1}{80} = 12.5e^{-3}$. If we look at the maximal time-step for $relTol = 10^{-6}$ in table 4.3 we see that it equals $8e-3$. This means that every pulse is taken. Furthermore we are talking about the maximal time-step and therefore we conclude that the activation function is really used by

[3]This is only true if the used step-sizes are very small compared with 0.001(the distance of the output points).



(a)



(b)

Figure 4.9: The tendon force vs. time. (a) The force plot belonging to the best found solution of experiment 1. Here we used pulses with $f = 40$. We honestly did not expect this shape. We expected a less or more constant force at the end. (b) The force plot that belongs to best found solution of experiment 2(f=80). The produced force is more "constant" at the end.

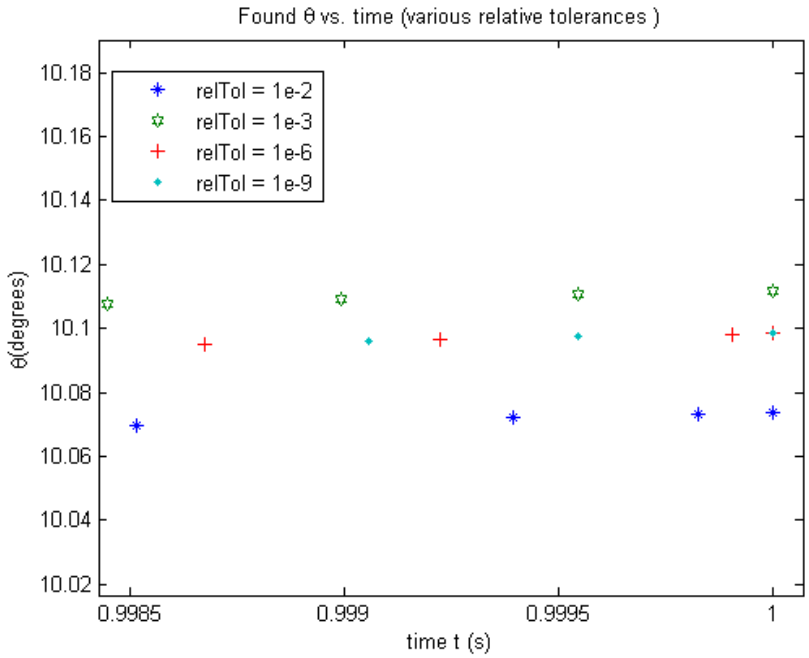


Figure 4.10: Plot of θ belonging to the best solution vs time t, solved with different relative tolerances $relTol = 1e^{-2}, 1e^{-3}, 1e^{-6}, 1e^{-9}$)(ODE45 solver). Here we plotted the calculated θ instead of the interpolated θ. We have zoomed in at $t \approx 1$ and the usage of variable time-steps is clearly visible. We assume the higher the relative tolerance the higher the accuracy. From the plot we can see that solving with a relative tolerance of 10^{-6} is pretty accurate. At the endpoint, $t = 1$, the error between the found θ with $relTol$ $1e - 6$ and $1e - 9$ equals $1.4e^{-005}$.

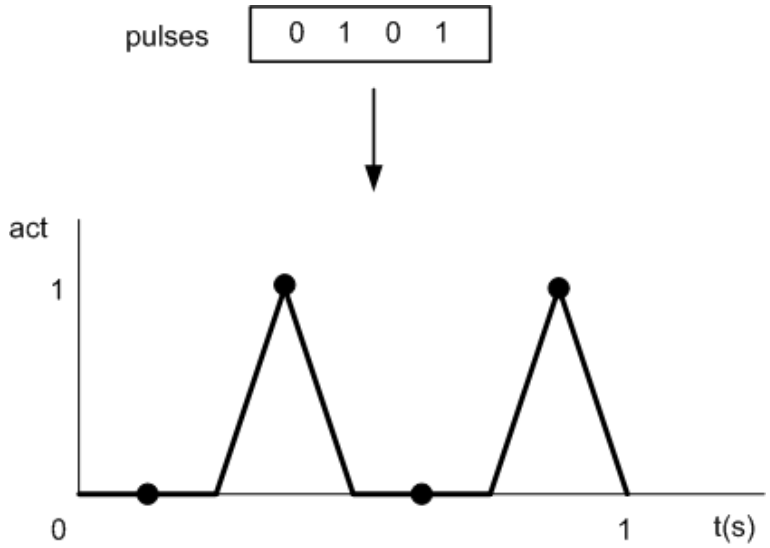


Figure 4.11: Triangular-shaped pulses instead of constant pulses. A triangular pulse has less activation than a constant pulse.

the model. Things change if we use other type of activation function that are 'finer'. Solutions with higher frequencies or recorded EMG data for example. If the taken time-steps are too large, pieces of the activation can be missed by the model. By specifying the maximal time-step manually, we can prevent this. The price for this is more computation time however.

4.7.4 Experiment 3: another type of activation function

For our activation functions we used piecewise constant functions that we call pulses. What if the pulses are not constant but of triangular shape(see figure 4.11)? We use our best solution(from experiment 2) and calculate the activation function with 4.5:

$$act(t) = \sum_{i=0}^{f-1} p(i)\alpha_i(t) \tag{4.5}$$

where

$$\alpha_i(t) = \begin{cases} 2(t - iw) & iw \leq t < (i + \frac{1}{2})w \\ 2 - 2(t - iw) & (i + \frac{1}{2})w \leq t \leq (i + 1)w \\ 0 & \text{elsewhere} \end{cases}$$

Then we used this activation function to simulate our thigh model. The solver is the well-known ODE45 with $relTol = 1e^{-6}$. The details of the simulation are:

- $step_{min} = 6e^{-18}$
- $step_{max} = 2e^{-2}$ (very large)

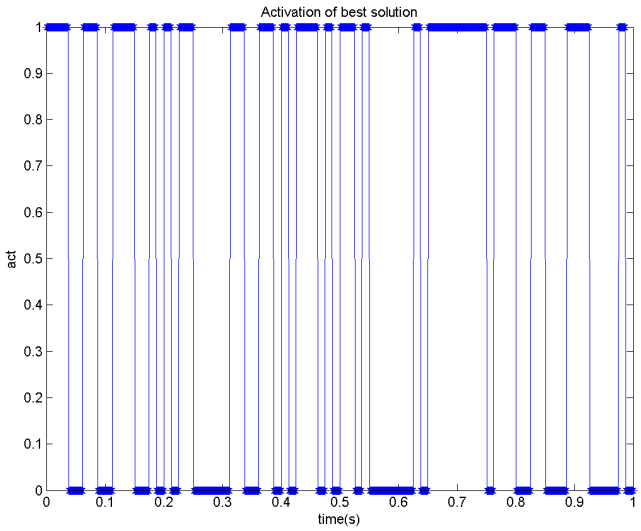
- $\theta_{max} \approx 0.1$ degree
- $\mathcal{F} = 3.8369$ (interpolation)
- $\mathcal{F}_2 = 4.9767$ (real output values)

There is a big difference between $\mathcal{F}$ and $\mathcal{F}_2$. This could be the result of the relative large $step_{max}$ [4]. We can prevent this by specifying $step_{max}$ manually. In figures 4.12 and 4.13 we have some interesting plots of the used activation and the found θ. Here we see that specifying $step_{max}$ is very important. The step-size used by the solver depends on the state of the system. The state of the system in turn depends on the activation. When relatively big time-steps are used parts of the activation will be missed. In this case the largest time-steps $step_{max} = 0.02$ were used. The width of a pulse is $w = 0.0125$ and some pulses will be missed. The simulation is done with a different activation than what we have in mind. This is a wise lesson for the future. We should specify $step_{max}$ small enough, otherwise another activation is used for the simulation.

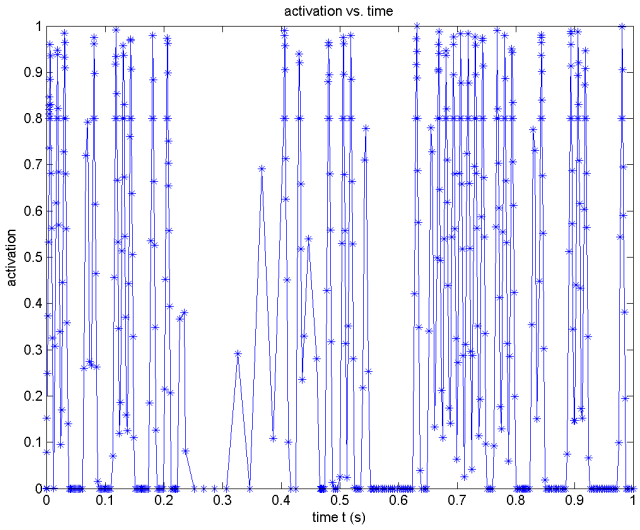
4.8 Conclusion

We have implemented an EA from scratch to solve optimization problems. If we want to use the EA for our model we should formulate the goal in terms of the output and encode the activation function into an array of variables.
We used the EA to solve a problem that can be seen as a control problem. The goal was to hold the thigh static for a moment of time. For this goal we formulated an objective function in terms of θ and used the EA to minimize this objective function.
Encoding the activation function was a bigger problem. In the ideal situation we can parameterize the activation, put these parameters into an array and use EA to find the optimal array of parameters. Because we do not how we can parameterize the activation function properly we used a simplified type of activation function. These functions are made up with piecewise constant functions that we called pulses. Real neural activation functions do not look like these functions.
With the simplified activation functions we get good results. The EA we implemented can also be used to solve other optimization problems.
After the experiments we realized that the approximated objectives can contain errors if the used step-sizes were too big. With very big step-size we not only have large interpolation errors, but also risk that other activation is used instead of the found activation. After a thorough examination with the best found solution we concluded that our approximated objectives are reliable. In another small experiment we used triangular pulses instead of constant pulses. Here we see that big step-sizes will produce unreliable objective approximations. We thus should not only focus on the relative tolerance but also pay attention to the maximal step-size.

[4] We used the default $step_{max}$, which is set to 1/50 of the simulation time. In our case it equals 1/50=0.02, the default $step_{max}$ is thus used.

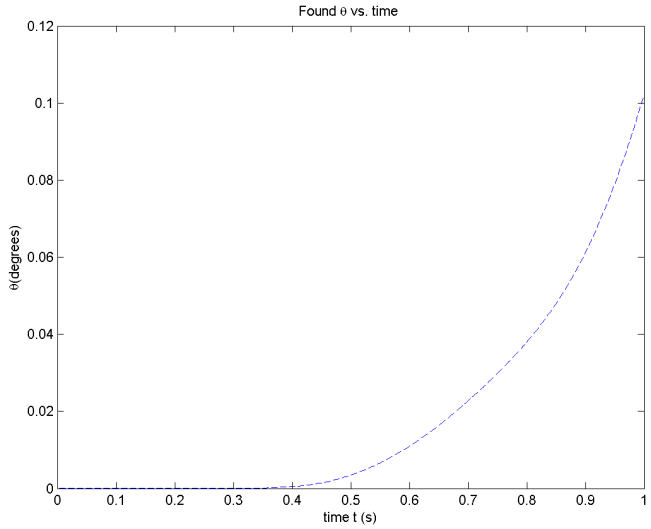


(a)



(b)

Figure 4.12: Result of experiment 3. (a) The activation function belonging to the best solution where the pulses are constant. This is a reference for the case that we use triangular instead of constant pulses. (b)The used activation vs time(variable time-steps). Here triangular pulsus are used. We can see that when large step sizes ($t \approx 0.3$) were used pulses are missed. The simulation is done with a "different" activation.



(a)

Figure 4.13: (c) Results of experiment 3(continued). The found θ vs. time. $\theta_{max} \approx 0.1$ degree. With triangular pulses we have less maximal activation and thus muscle force and we expected a smaller θ_{max}, but not this small. When the state of the model is less dynamic, larger time-steps are used. Non-zero pulses that cause the muscle to produce force risk being missed by the model.

Conclusion

We wondered if we could use the computer to find martial arts techniques. Martial arts techniques are optimal motions for a certain goal. We therefore have to model the locomotor or musculoskeletal system of the human body and perform optimizations on the model. The musculoskeletal system is very complex and can be split into the skeletal and the muscular system. Both are highly dynamic systems. To get results on screen we should write software. We have chosen Simulink, a platform for modelling and simulation of multi-domain dynamic systems, to implement the musculoskeletal system. Throughout the thesis it became very clear that using Simulink is about the easiest way to model a musculoskeletal system. Especially with the help of SimMechanics and Virtual Muscle 3.1.5.
Because of the complexity we modelled only parts of the musculoskeletal system. We started with the skeletal system first. The skeletal system is divided into several segments and rigid bodies were used to represent these segments. This model is also known as an articulated model and is a multi-body system. The complexity of our musculoskeletal system depends mainly on the size of the skeletal system. The more bodies it contains, the more muscle models should be added. We used SimMechanics to model the skeletal system. With SimMechanics it was very easy to model and extend the skeletal-system.
With the choice of modelling the skeletal system as a system of rigid bodies, we moved into the field of multi-body dynamics. Here the dynamics are well understood. In the world of muscle modelling things are different. Scientists still do no know how the muscle behaves precisely. We have chosen Virtual Muscle to model the behavior of muscles. Virtual Muscle is an empirical skeletal muscle model. It can predict the muscle's behavior at sub-maximal activation relatively well compared with other muscle models. Because the maximal voluntary activation of human muscles is still sub-maximal, we believe that Virtual Muscle is the right choice. Virtual Muscle treats the muscle as a scaled sarcomere and we can easily build realistic muscle models if we have good macroscopic muscle data. The if is a very big if, because the skeletal and muscle models should fit together.
Finally it was time to perform optimizations. We had implemented an evolutionary algorithm(EA) with a wide variety of operators from scratch. We used our EA to solve a problem that can be considered as a control problem. In that problem we have modelled a thigh that is driven by a skeletal muscle and wondered if the thigh could be kept still at a certain degree. Wether this is possible we did not know. The EA produced results that are close to what we want. A major drawback of using EA is the large number of simulations that is needed. A simulation is very costly, even for our simple thigh model. A way

to handle this problem is to implement the model more efficiently on another platform. The Simulink model can then be used to verify the program. A more rigorous way is to use more computers. The EA is well suited for trivial parallel programming.
Although we did not find a martial arts technique we have made a first step at how we can do that. During this thesis our respect for the human body and human mind has grown more and more. The human body as a piece of wonderful engineering and the human mind as a great wonder. One should keep both of them in good shape, by practicing martial arts for example.

Appendix A

How does Virtual Muscle work?

The equations that Virtual Muscle uses to describe the behavior of a mtc is given in chapter 3. While some are relatively easy to understand, others are not. With BuildMuscles we can export mtc models as Simulink block diagrams. From figure A.1 to figure A.6 we will explain how the output force of a mtc-block is calculated. We have built a mtc model that models the *rectus femoris*. We only used one unit to model the whole muscle group.

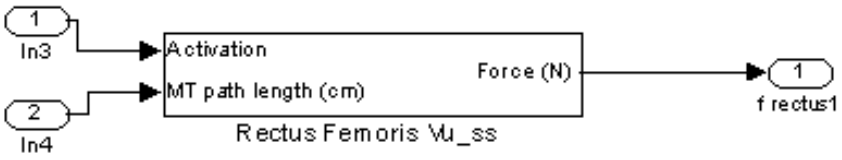


Figure A.1: The top-level view of a mtc-block. The input and output are clearly given.

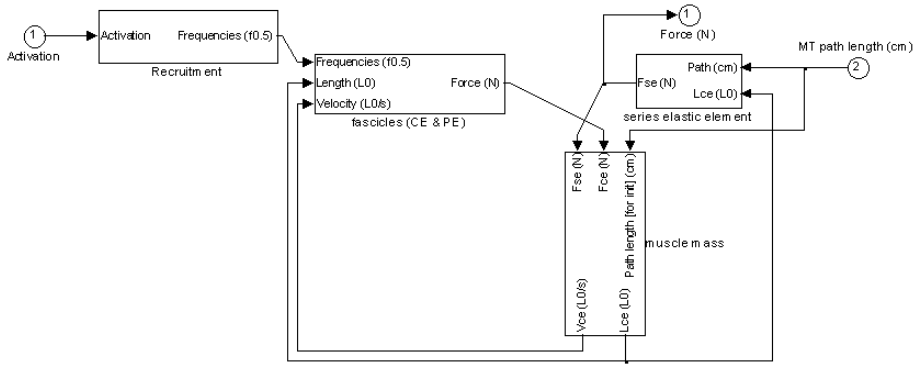


Figure A.2: View under the mtc-block. We have a recruitment block, a fascicle block, a serial elastic block and a mass block. The recruitment block turns the normalized activation into stimulation for each fascicle. The fascicle block calculates the fascicle force. The serial elastic block calculates the pulling force on the skeleton. The mass block keeps track of the fascicle length and contraction speed.

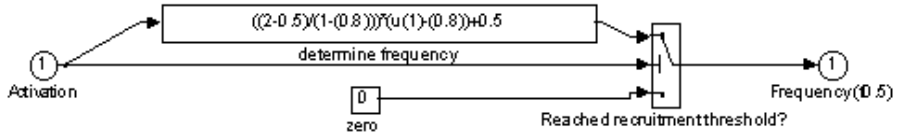


Figure A.3: The recruitment block. It models the recruitment of the muscle units. In this block the activation is turned into stimulation *stim* for each fascicle or muscle-unit. The numbering of input is done from top to bottom. The switch is set to recruitment threshold u_r If input 1 is larger than the recruitment threshold it will pass, else the value will be set to zero, so it will not be recruited.

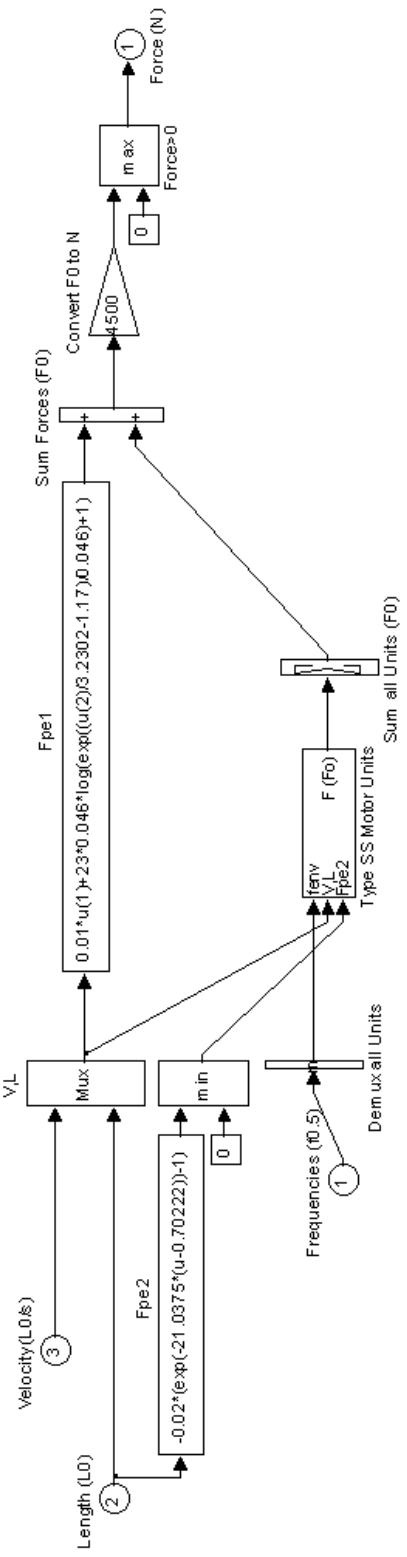


Figure A.4: The most complex block of the mtc-block. The PE1 block comes into action when the fascicle exceeds a certain length. The PE2 block models the thick filament compression. It comes into action when the fascicle decreases to a certain length. The CE block is fairly complex and we will look at it later on. The calculated force is first normalized to the optimal force F_0.

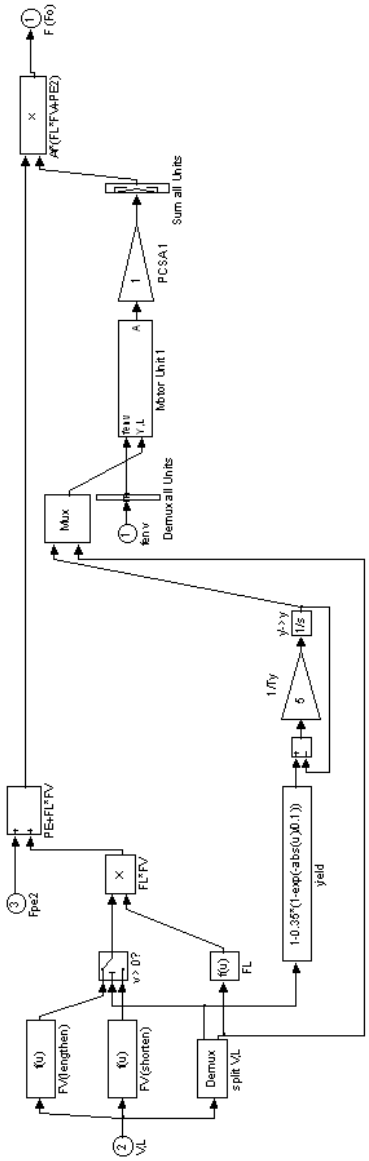


Figure A.5: The CE block revealed. It has (shortening length, shortening speed), Fpe2 and fenv(fire-frequency) as input and the normalized fascicle force as output. The CE force is a product of the outcome of the force-length, force-velocity and effective activation relation. The FL block models the force-length relationship. The shortening and the lengthening block models the force-velocity for concentric and eccentric contraction respectively. The effective activation relation is fairly complex and it takes diverse phenomena into account. Some of them are fiber-type dependent.

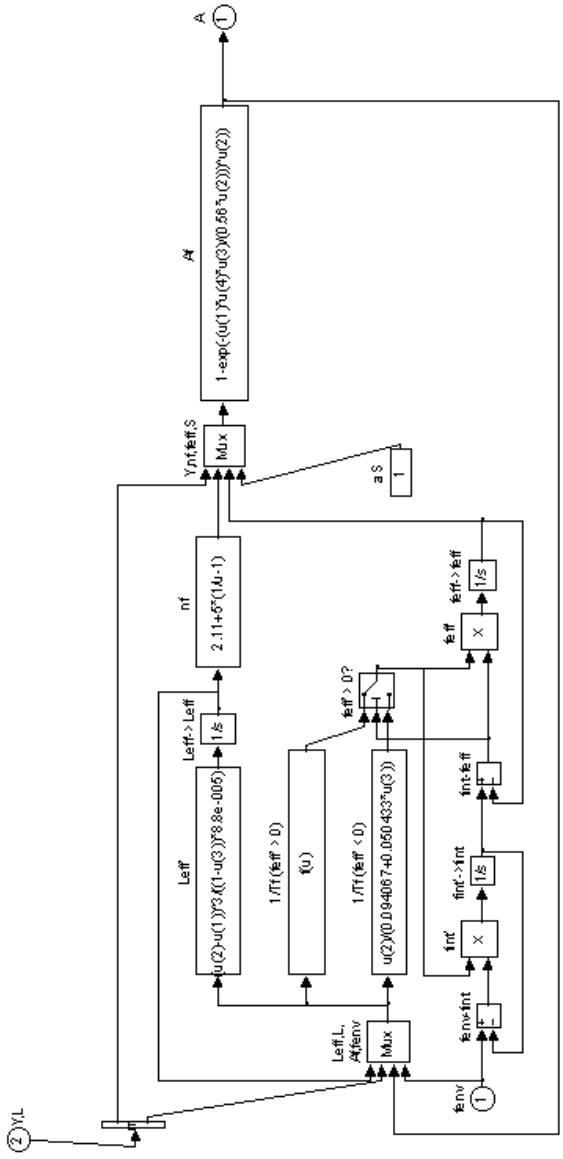


Figure A.6: The effective activation block. It outputs a value between 0 and 1, that represent the effective activation. This block takes all complex observed phenomena into account like yielding, sag, rise and fall times etc.

Appendix B

Grieve's method

B.1 Calculating the mtc-length with Grieve constants.

In order to compute the muscle-force we need to compute the length of the muscle-tendon complex(mtc) l_{oi}[1]. l_{oi} depends on the joint angles of the joints they cover. Mtcs that cover one joint are called *mono-articular* mtc and mtcs that cover 2 joints are called *bi-articular* mtc. In figure B.1 we see a sketch of how we can compute the l_{oi} of a mono-articular mtc. We call this the *radian approach* and we have

$$l_{oi}(\theta) = l_0 + r * \theta \tag{B.1}$$

where r is the radius of the joint and l_0 is the length of the mtc for $\theta = 0$. The way we computed l_{oi} however is with Grieve's method. Here the joint-l_{oi} relationship is measured via experiments. Then a second order polynomial is fitted. The found coefficients are called the Grieve constants.

$$l_{oi} = \sum_{i=0}^{j} A_{i,0} + A_{i,1} * \theta_i + A_{i,2} * \theta_i^2 \tag{B.2}$$

For mono-articular muscles we have $j = 0$. For bi-articular muscles we have $j = 1$. Grieve constants depend on the way θ is defined. Usually the null-state ($\theta = 0$) is when both segments are aligned. Notice that when $A2 = 0$, we have a similar equation like with the radian approach.

B.2 Calculating the moment-arm with Grieve constants.

Once the geometry of the joints and segments is known, we can determine the moment-arm. This can be rather complex, because each muscle has its own geometry. A more generic way to solve this problem is with virtual work. If we rotate the joint over an infinitesimal angle δ an amount of virtual work is done.

[1] o stands for origin and is closer to the trunk. i for insertion and is more away from the trunk

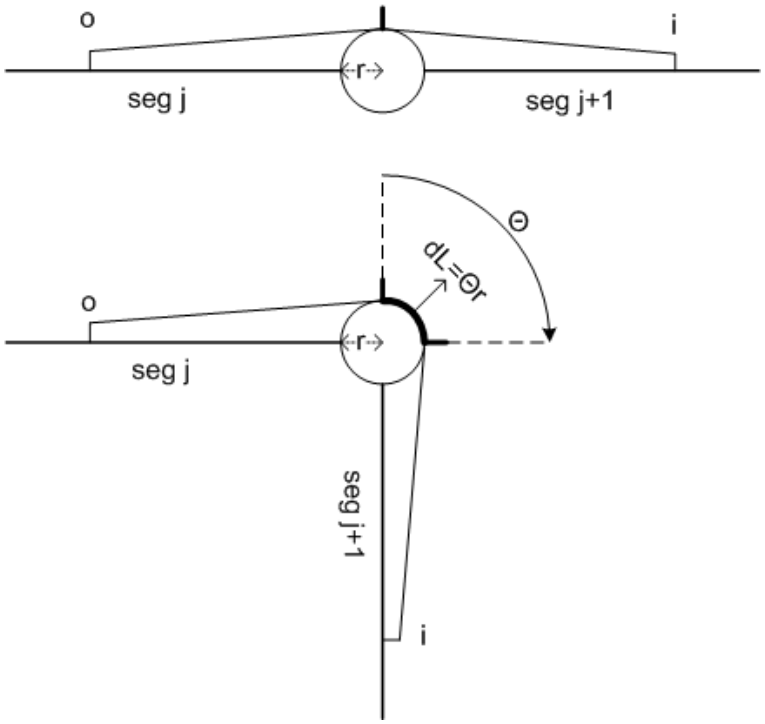


Figure B.1: An intuitive way to determine l_{oi}. Seg i and seg $i+1$ represent the midline of 2 segments. The circle represent the covered joint. On top we have the zero state, that is where θ is defined as zero. We assume that the MTC will follow the joint closely which leads to a radian increase. This is however not always the case. A clear example are the flexors of the knee. There we have a straight line crossing.

The work done to cause the rotation is then $\delta M = \delta a(\theta)F$. We also know that l_{oi} will change by the muscle force. The work done is given with $\delta l_{oi}\vec{F}$. They should be equal to each other, thus $\delta a(\theta)F = \delta l_{oi}\vec{F}$. Rearrangement leads to:

$$a(\theta) = \frac{\delta l_{oi}}{\delta \theta} \tag{B.3}$$

With infinitesimal small steps we have $a(\theta) \approx \frac{dl_{oi}(\theta)}{d\theta}$ With B.2 we have an equation for l_{oi}. Differentiation of this equation with respect to θ gives us the equation for the moment-arm:

$$a(\theta) = A_1 + A_2 * \theta \tag{B.4}$$

If A_2 is zero we have a constant moment-arm. Grieve constants are experimentally determined. The length of the mtc is measured at various joint angles and regression analysis is used to yield the Grieve constants(see B).

Appendix C

The GA-operators

For our EA we have implemented several operators and methods. We only list their names and their parameters. For more details we refer to [1]. The mutation and recombination operators depend on how we have encoded the solutions. The implemented operators are suited for multi valued variables, not for real valued variables.

C.1 Population Parameters

Population size N The number of solutions we look at every-time. If N is too small we risk convergence to local optima. If N is too big, we need too much computer resources.

Evolution time G Once a generation is created the evolution time increase with one. If G is small we risk we risk premature termination.

Streak s If G is very large, but there is no improvement for a number of generations it is likely that our EA has found the optimal solution or got stuck in a local optimum. Further searching can be a waste of computer resources. By specifying s the search is stopped, if there is improvement after s generations

C.1.1 Region Parameters

Region-size n the population is divided into regions. n is the number of solutions per region.

Isolation time g the regions will evolve independently from each other for a number of generations. This number is specified with g, the isolation time.

C.2 Initializing the population

The initialization is done with randomly generated solutions.

C.2.1 Terminal constraints

In our case the search space is enormous. Furthermore the costs for evaluating our fitness function are very expensive. We can stop the search based on the following criteria.

Goal When the goal is reached, the search can be stopped. This is an ideal situation. We should remark that we can not always define a goal.

Time (s) When the searching time exceeds a user specified amount of time, the search will be stopped.

Costs (fitness evals) A more objective stop criterion is the amount of fitness evaluations.

Streak There is a chance that the EA is stuck in a local optimum. Streak is the number of generations that the search does not improve.

C.2.2 Selection

For each generation we should select a number of members to produce offsprings.

Truncation(nmates): nmates is number of parents for the next generations. The minimum is 2.

Tournament(nmates, sp): sp is selection pressure, needed for linear ranking. The sp should be in the range of [1.0, 2.0].

Roulette(nmates, sp,size): size is the tournament size.

Linear ranking

With tournament and roulette selection fitter members have more chance to be selected. These selection chances are proportional to their fitness evaluation. Sometimes we have too dominant solutions. Their proportion is way bigger and it is very likely that weaker members have practically no chances to be selected. Think of 1 percent or less. Linear ranking gives weaker members relatively fair chances to be chosen too. This will guarantee the diversity. The *sp* parameter regulates the rate of diversity.

Elitism

With elitism we keep a number of best solutions of the current generation for the next generation. These are called the elite members. By doing this best solution of each generation will be better or equal to the best solution of the previous generation. Typical numbers of elite members are 1 or 2.

C.2.3 Recombination

The implemented recombination operators are:

discrete : Two children are created from two parents with the use of 2 random index-arrays.

uniform : Same as with *discrete*, but with one random index-array.

crossover(m): m is the number of cross-over points. If m is small (eg. 1 or 2), the children will inherit much of their parent's "look". This of course depends on how the solution is encoded.

C.2.4 Mutation

We have only implemented one type of mutation and that is mutation with binary valued variables. It has the following parameters:

Number of mutants: the number of children where mutation operator will be applied to.

Mutation rate p: each variable of a solution will mutate with chance p. The recommended value is $1/l$, where l is the number of variables of a solution.

Bibliography

[1] L.F. Shampine. *Numerical Solution of Ordinary Differential Equations.* Chapman & Hall. March 1, 1994

[2] *Simulink© 6 Using Simulink©.* The MathWorks, Inc. 2006. http://www.mathworks.com

[3] Grant R.Fowles, George L.Cassiday. *Analytical Mechanics, 6th edition.* Saunders College Publishing. 1990.

[4] *Forward Dynamics of Multibody Systems: A Recursice Hamiltonian approach.* Vrije Universiteit Brussel, Brussel. September 2005.

[5] M.A.Mc Conaill, J.V.Basmajian. *Muscles and Movements: a basis for human kinesiology.* Baltimore, Maryland 21202 USA. 1969.

[6] *SimMechanics For Use with Simulink©, User's Guide Version 2.* The MathWorks, Inc. Natick, MA. 2006. http://www.mathworks.com

[7] P. Montazemi, R. Davoodi. *Comparison of Dynamic Engines for Musculoskeletal Modeling Software MSMS.* A.M. Institute for Biomedical Engineering, University of Southern California. 2005.

[8] David A. Winter. *Biomechanics and motor control of human movement.* Hoboken, New Jersey. 2005.

[9] B.M. Nigg, W. Herzog. *Biomechanics of the Musculo-skeletal System, 2nd Edition.* John Wiley & Sons Ltd, West Sussex PO19 iUD, England. 1999

[10] Ernest J. Cheng , Ian E. Brown, Gerald E. Loeb. *Virtual muscle: a computational approach to understanding the effects of muscle properties on motor control.* Department of Biomedical Engineering, Alfred E. Mann Institute for Biomedical Engineering, Southern California, USA. June 2000.

[11] Ernest Cheng, Ian Brown, Jerry Loeb. *Virtual Muscle 3.1.5 Muscle Model for MATLAB User's Manual.* Feb 21, 2001.

[12] Jack M. Winters, Savio L-Y. Woo. *Multiple Muscle Systems Biomechanics and Movement Organization.* Springer-Verslag, New York. 1990

[13] Stuart I Fox. Pierce College. *Human Physiology, 8th edition.* McGraw-Hill Science/Engineering/Math. 2003.

[14] Hartmut Pohlheim. *GEATbx Introduction Evolutionary Algorithms: Overview, Methods and Operators version 3.7.* 2005. http://www.geatbx.com.

[15] Agamemnon Despopoulos, Stefan Silbernagl. *Color Atlas of Physiology, 5th edition.* Thieme Medical Publishers Georg Thieme Verlag, New York. 2003. http://www.thieme.com.

[16] Ian Edward Brown. *Measured and Modeled Properties of Mammalian Skeletal Muscle.* Queen's University.
Kingston, Ontario, Canada. 1999.