HuMAnS User Documentation

October 23, 2006

This documentation is under construction. The different parts are not finalized and the links between them do not exist yet.

Contents

Chapter 1

General Principles of the simulation in HuMAnS

1.1 General Principles of the simulation in HuMAnS

1.1.1 Dynamic modelisation of a free system

The dynamic modelisation of a free system can be set in the form:

$$M(q)\ddot{q} + N(q,\dot{q})\dot{q} + G(q) = T(q)u$$

where q is the system state variable, $M(q)$ the system inertia, $N(q,\dot{q})$ a non-linear effects matrix, $G(q)$ the forces differential of a potential (for example the gravity), u a command vector and $T(q)$ a matrix describing the effect of this command on the system.

Howewer, when a bipede is walking, it is supported by contact points and constraints due to these contacts occur. That is why a contact model must be developped in order to describe the dynamic of a system with these constraints.

1.1.2 Contact model

The contact model is based on the two following hypothesis, developped in the next paragraphs:

- the contact points cannot penetrate the ground or the environment;
- when a contact exists, the contact points do not slide on the ground or on the environment.

Unilateral constraints

The solids which compose the system cannot penetrate the ground or the environment. So we have a set of inegalities on the position of the model (the robot or the human), that is, on the state vector q:

$$\varphi_n(q) \geq 0$$

If we consider the points in contact with the environment at time t and if we indicate them with an $*$, we can write:

$$\varphi_n^*(q) = 0$$

with φ_n^* the set of active constraints at t. None of these points of contacts penetrate the ground, neither before nor after the time t. The velocity and acceleration of these points must be directed in order to avoid penetration. These constraints on velocities and accelerations are obtained by successive differentiation:

$$C_n(q)\dot{q} = 0 \tag{1.1}$$

$$C_n(q)\ddot{q} + s_n(q,\dot{q}) = 0 \tag{1.2}$$

with $C_n(q) = \partial\varphi_n^*/\partial q$ the jacobian of φ_n^* and $s_n(q, \dot{q})$ the other terms appearing in the differential.

For each enabled unilateral constraint, that is to say for each component of φ_n equal to zero, two bilateral constraints are added in order to prevent the sliding of the corresponding contact point. Naturally, these two constraints are disabled when the contact is broken.

Bilateral constraints

We suppose that points in contact with the environment do not slide on it. Then they do not move directions parallel with the ground or environment. So, we have constraints on the state vector q:

$$\varphi_t^*(q) = 0$$

and on the velocity and the acceleration of the system which are obtained by successive differentiation:

$$C_t(q)\dot{q} = 0 \tag{1.3}$$

$$C_t(q)\ddot{q} + s_t(q, \dot{q}) = 0 \tag{1.4}$$

with $C_t(q) = \partial\varphi_t^*/\partial q$ the jacobian of the non-sliding constraints and $s_t(q, \dot{q})$ the other terms appearing in the differential.

1.1.3 Dynamic with contacts

We can write the system dynamic with contact in the form:

$$\begin{cases} M(q)\ddot{q} + N(q, \dot{q})\dot{q} + G(q) = T(q)u + C_n(q)^T\lambda_n + C_t(q)^T\lambda_t \\ \varphi_n(q) \geq 0 \\ \varphi_t^*(q) = 0 \end{cases} \tag{1.5}$$

with $C_n(q)^T\lambda_n$ the interaction forces due to the unilateral constraints $\varphi_n(q)$ and $C_t(q)^T\lambda_t$ thoses due to the bilateral constraints $\varphi_t^*(q)$.

1.1.4 Impacts

Such a dynamic system generates impacts involving discontinuities of the velocity which follow the law below:

$$\begin{cases} M(q)(\dot{q}_+ - \dot{q}_-) = C_n(q)^T\Lambda_n + C_t(q)^T\Lambda \\ C_n(q)\dot{q}_+ = 0 \\ C_t^(q)\dot{q}_+ = 0 \end{cases} \tag{1.6}$$

this law connects the velocity after impact $\dot{q}_+$ to the one before impact $\dot{q}_-$ under the impulsive forces $C_n(q)^T\Lambda_n$ and $C_t(q)^T\Lambda_t$ where $C_n(q)$ and $C_t(q)$ are the jacobian of unilateral and bilateral enabled constraints at the impact time. This law does not allow unilateral constraints to break during an impact, that is not always correct but that was necessary to avoid the appearance of points of impacts accumulation which would prevent any simulation.

1.1.5 Quadratic Problems used in HuMAnS

The dynamic system (1.5) can be set in the form of a quadratic problem:

$$\begin{cases} \min\limits_{\ddot{q}} \quad \frac{1}{2}\,\ddot{q}^T M(q)\ddot{q} + \ddot{q}^T[N(q,\dot{q})\dot{q} + G(q) - T(q)u] \\ C_n(q)\ddot{q} + s_n(q,\dot{q}) \geq 0 \\ C_t(q)\ddot{q} + s_t(q,\dot{q}) = 0 \end{cases} \tag{1.7}$$

And in a same way, the impact law can be set in the form:

$$\begin{cases} \min\limits_{\dot{q}_+} \quad \frac{1}{2}\,\dot{q}_+^T M(q)\dot{q}_+ - \dot{q}_+^T M(q)\dot{q}_- \\ C_n(q)\dot{q}_+ = 0 \\ C_t(q)\ddot{q}_+ = 0 \end{cases} \tag{1.8}$$

These two quadratic problems are used during the simulation to compute at each time the acceleration and velocity of the system after impact.

1.2 Muscle modelisation in HuMAnS

1.2.1 Command signal

The command signal $u(t)$ is shown on the figure 1.1. It fluctuates between two values U_p and U_m. Its different states correspond to different activations:

- After a stimulation, the command signal jump to state 2 for a period τ_1 which correspond to the stimulation time. It's value is U_p during this phasis;
- At the first tick of clock, the command signal enters in a relaxation phasis of duration τ_2. This state is called 1 in HuMAnS;
- When the impulse stops (*ie* at the second tick of clock), the command signal falls in state 0 in which it is equal to U_m. When an impulse is sent again, a delay τ elapses it before the signal command get back again in state 2.

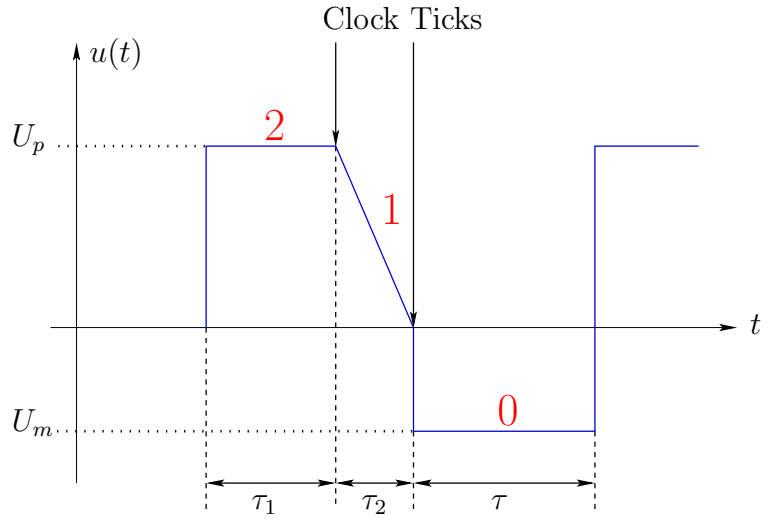


Figure 1.1: Command signal. The number corresponds to the signal state numbering.

The U_p, U_m, τ, τ_1 and τ_2 values used in HuMAnS are given in the table 1.1. This values are stocked in the **DefaultConstants.sci** files of the `ActuationModel/StaticCalciumKneeMuscles` and `ActuationModel/StaticCalciumRightKneeMuscles` modules.

1.2.2 Recruiting rate

...laius sur ce que c'est...

U_p	5 s^{-1}
U_m	5 s^{-1}
τ	50 ms
τ_1	20 ms
tau_2	10 ms

Table 1.1: Parameters values of command signal.

The recruiting rate depends on the width and intensity of the impulsion. Its value is given by:

$$\alpha = \frac{1}{2} + \frac{1}{2} \frac{\tanh(\ 5\ (\frac{pwi}{pw_{max} i_{max}} - \frac{1}{2}\ ))}{\tanh(\ \frac{5}{2}\ )}$$

with pw and i the width and the intensity of the impulse and pw_{max} and i_{max} the maximum width and intensity of the impulse. These maxima are defined in the **DefaultConstants.sci** files of the **ActuationModel/StaticCalciumKneeMuscles** and **ActuationModel/StaticCalciumRig** modules. Their values are given by the table 1.2

pw_{max}	1.4 ms
i_{max}	200 mA

Table 1.2: Maximum width and intensity of an impulse.

1.2.3 Muscle model and Force-Length relation

The muscle model used in HuMAnS is described on the figure 1.2.

The model considers that the muscle is composed of:

- a contractile component stimulated by the command signal $u(t)$. We note k_c its variable stiffness, F_c its force and L_{c0} its lenght at rest.
- a series linear spring which stiffness is k_s;
- and a parallel exponential spring which rest lenght is L_0 and which stiffness is ....????k_p et k_ep ??

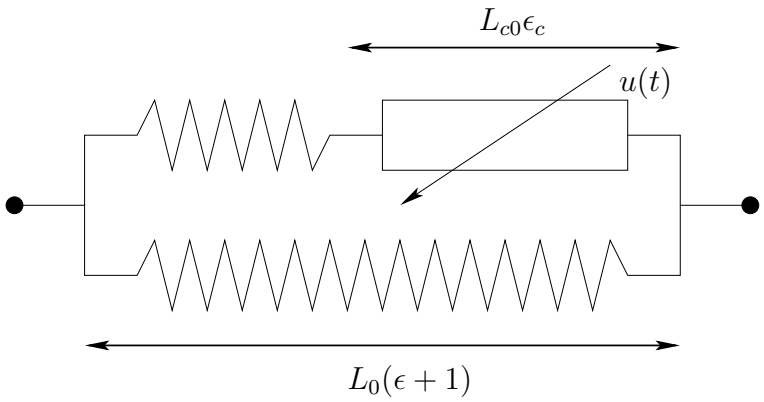


Figure 1.2: Muscle model

The stiffness and the force of a muscle depends on its length:

$$k_0 = k_m(1 - (\frac{\epsilon}{0.4})^2) \quad (1.9)$$

$$F_0 = F_m(1 - (\frac{\epsilon}{0.4})^2) \quad (1.10)$$

for $1.6 \leq \epsilon \leq 2.4$.

For the thigh muscles (the quadriceps and ischio), the k_m, F_m, k_s, k_p, k_{ep}, L_{c0} and L_0 values are given in the table 1.3. These values are defined in the **DefaultConstants.sci** files of the **ActuationModel/StaticCalciumKneeMuscles** or **ActuationModel/StaticCalciumRightKneeMusc** modules.

	quadriceps	*ischio*
k_m	$10^4\ N.m^{-1}$	$10^4\ N.m^{-1}\ ms$
F_m	$500\ N$	$500\ N$
k_s	$10^4\ N.m^{-1}$	$10^4\ N.m^{-1}\ ms$
k_p	$0\ N.m^{-1}$	$0\ N.m^{-1}\ ms$
k_{ep}	$1\ N.m^{-1}$	$1\ N.m^{-1}\ ms$
L_{c0}	$8.2\ cm$	$10.7\ cm$
L_0	$49.2\ cm$	$49.2\ cm$

Table 1.3: Maximum stiffness and force of quadriceps and ischio muscles

1.2.4 Muscle dynamic

Let ϵ_c be the elongation of the contractile component of the figure 1.2. The dynamic model of contractile component coupled with the linear series spring is:

$$\begin{cases} \dot{k}_c = -(|u| + |\dot{\epsilon}_c|)k_c + \alpha k_0 |u|_+ \\ \dot{F}_c = -(|u| + |\dot{\epsilon}_c|)F_c + \alpha F_0 |u|_+ + k_c L_{c0} \dot{\epsilon}_c \\ \dot{F}_c = k_s(L_0 \dot{\epsilon} - L_{c0} \dot{\epsilon}_c) \end{cases} \tag{1.11}$$

where $\dot{k}_c$, $\dot{F}_c$ and $\dot{\epsilon}_c$ are the unknown variables.

The relation between $\dot{F}_c$ and $\dot{\epsilon}_c$ in the last two equations consist of a straight line and of two straight half line. These straight lines are represented on the figure 1.3.

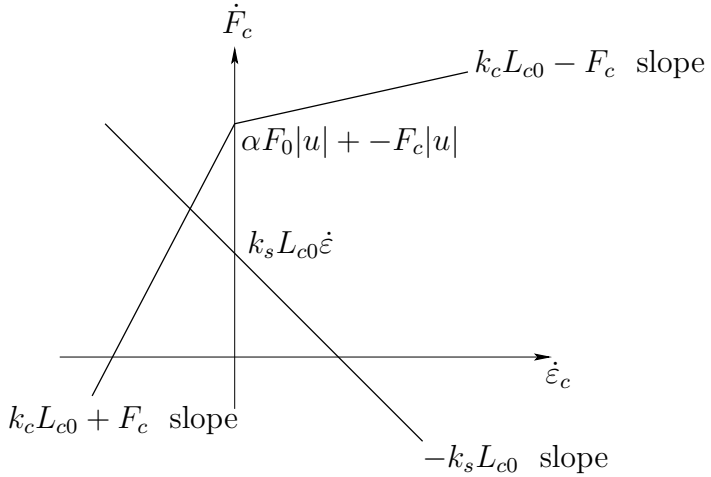


Figure 1.3: computation of the dynamics

In order to have one and only one solution to the system (1.11), the slope of the second half-line (for $\dot{\epsilon}_c \geq 0$) must be strictly greater than the slope of the straight line. So, we must verify that:

$$k_s L_{c0} + k_c L_{c0} - F_c > 0.$$

Then, we can compute $\dot{\epsilon}_c$ from the command u, the recruiting rate α, the forces F_c and F_0, the stiffness k_c and the differential of elongation $\dot{\epsilon}$ with the following equation:

$$\dot{\epsilon}_c = \frac{k_s L_0 \dot{\epsilon} + F_c |u| - \alpha F_0 |u|_+}{k_s L_{c0} + k_c L_c 0 - S_{\dot{\epsilon}_c} F_c}$$

with $S_{\dot{\epsilon}_c}$ the sign of $\dot{\epsilon}_c$. Then we can compute $\dot{k}_c$ and $\dot{F}_c$ with the first and last equations of the system (1.11).

The total force developped by the muscle is:

$$F = F_c + \frac{k_p}{k_{ep}} \left(e^{k_{ep}\epsilon} - 1\right)$$

with the second part of the right hand side corresponding to the force developped by the parallel exponential spring.

1.2.5 Knee articulation

In order to compute $\dot{\epsilon}_c$, we must compute the differential of the elongation of the parallel exponential spring $\dot{\epsilon}$. This differential is given by the geometric modelisation of the knee which is shown on the figure 1.4.

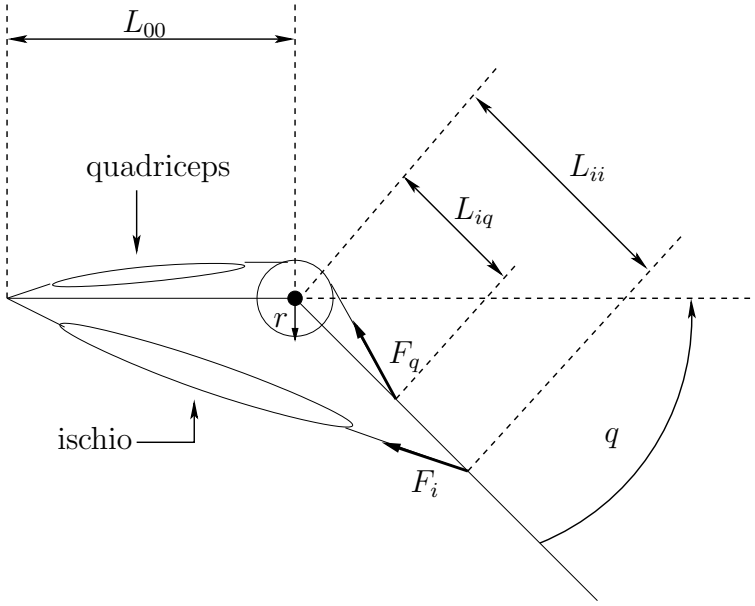


Figure 1.4: Geometric modelisation of the knee articulation

This geometric modelisation gives us:

$$\begin{cases} L_{quadriceps} &= rq + \sqrt{L_{00}^2 - r^2} + \sqrt{L_{iq}^2 - r^2} + r\sin^{-1}(\frac{r}{L_{00}}) + r\sin^{-1}(\frac{r}{L_{iq}}) \\ \epsilon_{ischio} &= \sqrt{L_{00}^2 + L_{ii}^2 + 2L_{00}L_{ii}\cos(q)} \\ \dot{\epsilon}_{quadriceps} &= r\dot{q} \\ \dot{\epsilon}_{ischio} &= -\frac{L_{00}L_{ii}\sin(q)}{\sqrt{L_{00}^2 + L_{ii}^2 + 2L_{00}L_{ii}\cos(q)}}\dot{q} \end{cases} \tag{1.12}$$

and we have:

$$\begin{aligned} L_{quadriceps} &= L_{0_{quadriceps}}(\epsilon_{quadriceps} + 1) \\ L_{ischio} &= L_{0_{ischio}}(\epsilon_{ischio} + 1) \end{aligned} \tag{1.13}$$

Then we can compute from the muscular forces the torque applied on the knee axis:

$$Torques = rF_q - \frac{L_{00}L_{ii}\sin(q)}{\sqrt{L_{00}^2 + L_{ii}^2 + 2L_{00}L_{ii}\cos(q)}}F_i - F_v\dot{q}$$

...donnees r, L00...

Chapter 2

Computation of Lagrangian Model

In this chapter, the general way to construct Lagrangian Models in the HuMAnS toolbox is explained.

The Lagrangian Models already put in the HuMAnS toolbox are placed in a directory name after the model under the `LagrangianModel` directory. The existing models are:

- the **Bip** model, in the `LagrangianModel/Bip` directory;
- the **Human** model, which joints and lengths are the same that for the **Bip** model, is placed in the `LagrangianModel/Human` directory;
- the **Human36** model which is a biomechanical model with 42 degrees of freedom and with anthropometric lengths, is placed in the `LagrangianModel/Human36` directory.

It has been shown in section 1.1.5 that the dynamic system can be set in the form of a quadratic problem:

$$\begin{cases} \min\limits_{\ddot{q}} \quad \frac{1}{2}\,\ddot{q}^T M(q)\ddot{q} + \ddot{q}^T[N(q,\dot{q})\dot{q} + G(q) - T(q)u] \\ C_n(q)\ddot{q} + s_n(q,\dot{q}) \geq 0 \\ C_t(q)\ddot{q} + s_t(q,\dot{q}) = 0 \end{cases} \tag{2.1}$$

where q is the state variable of the system, $M(q)$ the inertia, $N(q,\dot{q})$ a non-linear effects matrix, $G(q)$ the forces differential of a potential (for example the gravity), u a command vector, $T(q)$ a matrix describing the effects of this command on the system, $C_n(q)$ and $C_t(q)$ the jacobians of respectively non-penetrating and non-sliding constraints and $s_n(q,\dot{q})$ and $s_t(q,\dot{q})$, the other terms appearing in the differential of these constraints (see section 1.1.2).

Defining the Lagrangian Model mean computing these different vectors and matrices. The functions which allows to compute them are C-functions generated by maple programs. These C-functions are placed in the `LagrangianModel/ModelName` directory, where `ModelName` is the name of the considered model. The maple files allowing to generate these C-files are placed in the corresponding `LagrangianModel/ModelName/MapleCodeGeneration` directory. Maple files are always considered to be in this directory.

First of all, we will see how to compute the kinematic model. Then, the computation of the dynamical model will be explained. Finally, we will define the files allowing the creation of a vrml animation corresponding to a simulation.

2.1 Kinematic Model

In this section, we will first define the two ways implemented in HuMAnS to specify the geometric modelisation of a jointed system. Then, we will see the model of the tags (which are a set of characteristic points or markers on the model). Finally, the way to compute the different vectors and matrices describing the contact model is explained.

2.1.1 Geometric modelisation of a jointed system

In this section, we will see the two ways implemented in HuMAnS to compute the position of the jointed model, that is to say the position and orientation of each solid which compose it. The state vector q specifying the position and the orientation of one of these solids and the articular positions is sufficient to describe the position of the articular model. This modelisation is implemented in the **KinematicData.maple** files.

The first way consist of using the modified Denavit-Hartenberg. The second one consist of using the Cardan angles.

Modified Denavit-Hartenberg parameters

We used the Denavit-Hartenberg notation modified by Khalil and Kleinfinger for the **Bip** and the **Human** models.

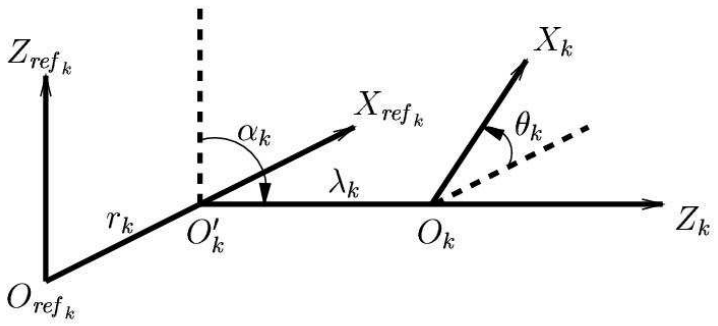


Figure 2.1: The Khalil and Kleinfinger representation describes the transformation from the ref_k frame to the k frame by making a translation r_k and a rotation α_k along X_{ref_k} axis, and then a translation λ_k and a rotation θ_k along Z_k axis.

An (O_k, X_k, Y_k, Z_k) frame is attached to each solid k. The position and the orientation of the frame k is given in the frame ref_k of the solid it is attached to. To described this position

and orientations, four parameters are used corresponding to a translation r_k and a rotation α_k along X_{ref_k} axis, and then a translation λ_k and a rotation θ_k along Z_k axis (figure 2.1). So the matrix which allows us to pass from a frame to an other is (in homogeneous coordinates):

$$\begin{pmatrix} \cos\theta_k & -\sin\theta_k & 0 & r_k \\ \cos\alpha_k \sin\theta_k & \cos\alpha_k \cos\theta_k & -\sin\alpha_k & -\lambda_k \sin\alpha_k \\ \sin\alpha_k \sin\theta_k & \sin\alpha_k \cos\theta_k & \cos\alpha_k & \lambda_k \cos\alpha_k \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Describing the position and orientation of an jointed model amounts to specify each of the parameters r_k, α_k, λ_k and θ_k from the geometry of the solids which compose the model and from the vector q components. The figure 2.2 shows a part of the **KinematicData.maple** file in which these parameters are specified.

```
# Nombre de solides
NSOL := 17+6:

# Nombre de degrés de liberté
NDDL := 15+6:

# Définition des coordonnées et vitesses généralisées
q := vector(NDDL):
qdot := vector(NDDL):

# Repère 1 : translation Y haut
ref_1    := 0:
r_1      := 0:
lambda_1 := 0.903+q[17]:
alpha_1  := -Pi/2:
theta_1  := 0:

# Repère 2 : translation Z droite
ref_2    := 1:
r_2      := 0:
lambda_2 := q[18]:
alpha_2  := Pi/2:
theta_2  := Pi/2:

# Repère 3 : translation X avant
ref_3    := 2:
r_3      := 0:
lambda_3 := q[16]:
alpha_3  := Pi/2:
theta_3  := Pi/2:

# Repère 4 : rotation Y lacet
ref_4    := 3:
r_4      := 0:
lambda_4 := 0:
alpha_4  := Pi/2:
theta_4  := Pi/2+q[20]:

# Repère 5 : rotation Z tangage
ref_5    := 4:
r_5      := 0:
lambda_5 := 0:
alpha_5  := Pi/2:
theta_5  := Pi/2+q[21]:
.
.
.
.
.
.
# Repère 18 : cardan gauche
ref_18    := 17:
r_18      := 0.410:
lambda_18 := 0:
alpha_18  := 0:
theta_18  := -q[6]:

# Repère 19 : pied gauche
ref_19    := 18:
r_19      := 0:
lambda_19 := 0:
alpha_19  := -Pi/2:
theta_19  := q[5]:

# Repère 20 : (inexistant)
ref_20    := 0:
r_20      := 0:
lambda_20 := 0:
alpha_20  := 0:
theta_20  := 0:

# Repère 21 : levier lombaires
ref_21    := 6:
r_21      := 0:
lambda_21 := 0.128:
alpha_21  := Pi/2:
theta_21  := Pi+q[13]:

# Repère 22 : cardan armoire
ref_22    := 21:
r_22      := 0:
lambda_22 := 0:
alpha_22  := Pi/2:
theta_22  := Pi/2+q[14]:

# Repère 23 : support armoire + armoire
ref_23    := 22:
r_23      := 0:
lambda_23 := 0:
alpha_23  := -Pi/2:
theta_23  := q[15]:
```

Figure 2.2: Part of **KinematicData.maple** file which describes the position and the orientation of a jointed model with the modified Denavit-Hartenberg parameters.

Cardan angles or pitch-roll-yaw angles

We used the Cardan representation for the **Human36** model.

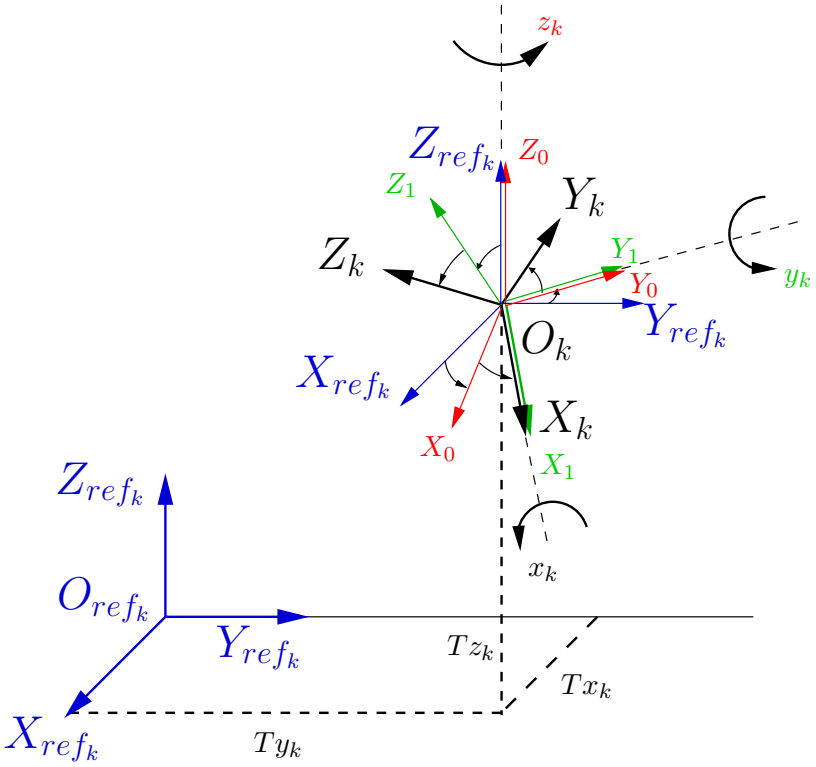


Figure 2.3: The yaw-pitch-roll or Cardan representation describes the transformation from the ref_k frame to the k frame by making a translation (Tx_k, Ty_k, Tz_k) and then a rotation of z_k around z_{ref_k} (roll), a rotation of y_k (pitch) around the resulting y-axis, which is the Y_0-axis, and finally a rotation of x_k (yaw) around the resulting x-axis, which is the X_1-axis

An (O_k, X_k, Y_k, Z_k) frame is attached to each solid k. The position and the orientation of the frame k is given in the frame ref_k of the solid it is attached to. To described this position and orientations, six parameters are used corresponding to a translation (Tx_k, Ty_k, Tz_k) and then a rotation of z_k around z_{ref_k} (roll), a rotation of y_k around the resulting y-axis (pitch) and finally a rotation of x_k around the resulting x-axis (yaw) (see figure 2.3). So the matrix which

allows to pass from a frame to an other is (in homogeneous coordinates):

$$\begin{pmatrix} \cos z_k \cos y_k & \cos z_k \sin y_k \sin x_k - \sin z_k \cos x_k & \cos z_k \sin y_k \cos x_k + \sin z_k \sin x_k & Tx_k \\ \sin z_k \cos y_k & \sin z_k \sin y_k \sin x_k + \cos z_k \cos x_k & \sin z_k \sin y_k \cos x_k - \cos z_k \sin x_k & Ty_k \\ -\sin y_k & \cos y_k \sin x_k & \cos y_k \cos x_k & Tz_k \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Describing the position and orientation of a jointed model amounts to specify each parameters x_k, y_k, z_k, Tx_k, Ty_k and Tz_k from the geometry of the solids which compose the model and from the vector q components. The figure 2.4 shows a part of the **KinematicData.maple** file in which these parameters are specified.

```
#solids number
NSOL := 17:

#degrees of freedom number
NDDL := 36+6:

#measured lengths number
NLANAT := 31:

#generalized coordinates and speed definition
q := vector(NDDL):
qdot := vector(NDDL):

#anatomical lengths definitions
L := vector(NLANAT):

# Frame 1 : LPT: Orientation and position
ref_1 := 0:
z_1 := q[42]:
y_1 := q[41]:
x_1 := q[40]:
Tx_1 := q[37]:
Ty_1 := L[14] + L[15] + L[16]+ q[38]:
Tz_1 := q[39]:

# Frame 2 : right thigh
ref_2 := 1:
z_2 := q[6]:
y_2 := -q[5]:
x_2 := -q[4]:
Tx_2 := 0:
Ty_2 := 0:
Tz_2 := L[3]/2:

# Frame 3 : right shank
ref_3 := 2:
z_3 := q[3]:
y_3 := 0:
x_3 := 0:
Tx_3 := 0:
Ty_3 := -L[6]:
Tz_3 := 0:

.
.
.
.
.
.
# Frame 14 : left humerus
ref_14 := 13:
z_14 := q[29]:
y_14 := q[28]:
x_14 := q[27]:
Tx_14 := -L[22]:
Ty_14 := L[20]:
Tz_14 := -L[21]:

# Frame 15 : left forearm
ref_15 := 14:
z_15 := q[31]:
y_15 := q[30]:
x_15 := 0:
Tx_15 := 0:
Ty_15 := -L[23]:
Tz_15 := 0:

# Frame 16 : left hand
ref_16 := 15:
z_16 := q[33]:
y_16 := 0:
x_16 := q[32]:
Tx_16 := 0:
Ty_16 := -L[25]:
Tz_16 := 0:

# Frame 17 : head
ref_17 := 8:
z_17 := q[36]:
y_17 := q[35]:
x_17 := q[34]:
Tx_17 := L[2]:
Ty_17 := L[26]:
Tz_17 := 0:
```

Figure 2.4: Part of **KinematicData.maple** file which describes the position and the orientation of a jointed model with the yaw-pitch-roll angles.

2.1.2 Tag model: the Tags.c file

Some characteristic points were put on the model. These points are called *tags*. Their specification is made in the **AdditionnalData.maple** file. For each tag, the user must give the number of the segment it is attached to, that is to say the number of the segment relative to which the tag does not move (`reftag_i` in the figure 2.5), and its position in the frame attached to this segment (`tag_i` in the figure 2.5). The figure 2.5 shows an example of an **AdditionnalData.maple** file (which is the **AdditionnalData.maple** file of the **Bip** model).

```
# Definition de quelques points importants (tags)

# Points de contact choisis parmi les tags
points_contact := [1, 2, 3, 4, 11, 12, 13, 14, 22, 23, 24, 25]:

#Definition des solides de contact
ContactSolids := matrix([[1, 4], [5, 8], [9, 12]]):

#Definition nombre de solides de contact
NCONTSOL := 3:

#Definition Nombre de Contacts
NCONT := 12:

#Definition vecteur Lambda:
Lambda := vector(3*NCONT):

# Nombre de tags
NTAG := 19:

# Tag 1 : pied droit
reftag_1 := 12:
tag_1    := vector([83, -90, 170]*1e-3):

# Tag 2 : pied droit
reftag_2 := 12:
tag_2    := vector([83, 90, 170]*1e-3):

# Tag 3 : pied droit
reftag_3 := 12:
tag_3    := vector([83, -60, -120]*1e-3):

# Tag 4 : pied droit
reftag_4 := 12:
tag_4    := vector([83, 60, -120]*1e-3):

# Tag 5 : cheville droite
reftag_5 := 11:
tag_5    := vector([0, 0, 0]*1e-3):

# Tag 6 : genou droit
reftag_6 := 10:
tag_6    := vector([0, 0, 0]*1e-3):
.
.
.
.
.
.
# Tag 15 : lombaires
reftag_15 := 21:
tag_15    := vector([0, 0, 0]*1e-3):

# Tag 16 : tronc
reftag_16 := 23:
tag_16    := vector([0, 0, 110]*1e-3):

# Tag 17 : tronc
reftag_17 := 23:
tag_17    := vector([0, 0, -110]*1e-3):

# Tag 18 : tronc
reftag_18 := 23:
tag_18    := vector([660, 0, 110]*1e-3):

# Tag 19 : tronc
reftag_19 := 23:
tag_19    := vector([660, 0, -110]*1e-3):

# Tag 20 : centre pied droit
reftag_20 := 12:
tag_20    := vector([83, 0, 0]*1e-3):

# Tag 21 : centre pied gauche
reftag_21 := 19:
tag_21    := vector([83, 0, 0]*1e-3):

# Tag 22 : attache tronc
reftag_22 := 23:
tag_22    := vector([660, 150, 110]*1e-3):

# Tag 23 : attache tronc
reftag_23 := 23:
tag_23    := vector([660, 150, -110]*1e-3):

# Tag 24 : attache tronc
reftag_24 := 23:
tag_24    := vector([660, -150, -110]*1e-3):

# Tag 25 : attache tronc
reftag_25 := 23:
tag_25    := vector([660, -150, 110]*1e-3):
```

Figure 2.5: Part of **AdditionnalData.maple** file which describes the positions of the tags in their attached segment frame.

The **ModelGeneration.maple** file generates the **Tags.c** file. After the linking in Scilab of this C-function, the user can call the **Tags** function with the state vector q in input and a matrix T of $(NumberOfTags + 1)$ rows and 3 columns on output. $NumberOfTags$ is the number of tags in the model. The row i ($i \leq NumberOfTags$) of this matrix is the position (x, y, z) of the tag i in the reference frame. The last row (the $(NumberOfTags + 1)$ row) is

the position of the model global center of mass. Then, T is on the form:

$$T = \begin{pmatrix} x^0_{tag_1} & y^0_{tag_1} & z^0_{tag_1} \\ x^0_{tag_2} & y^0_{tag_2} & z^0_{tag_2} \\ \vdots & \vdots & \vdots \\ x^0_{tag_{NumberOfTags}} & y^0_{tag_{NumberOfTags}} & z^0_{tag_{NumberOfTags}} \\ x^0_{CenterOfMass} & y^0_{CenterOfMass} & z^0_{CenterOfMass} \end{pmatrix}$$

where the 0 represents the reference frame.

The call of the **Tags** function in Scilab is in the following form:

```
> T = Tags(q);
```

2.1.3 Contact model

Contact vector: the Contact.c file

The **ModelGeneration.maple** file contains the **ContactVector** function which returns a vector composed of three parts of same size:

$$\begin{pmatrix} \varphi^1_t(q) \\ \varphi_n(q) \\ \varphi^2_t(q) \end{pmatrix}$$

These parts are used to specify the unilateral and bilateral constraints which appear in the dynamic:

$$\begin{array}{lcll} \varphi_n(q) & \geq & 0 & \text{(unilateral constraints)} \\ \varphi^*_t(q) & = & \begin{pmatrix} \varphi^1_t(q) \\ \varphi^2_t(q) \end{pmatrix} - Constant = 0 & \text{(bilateral constraints)} \end{array}$$

The * symbol indicates that these bilateral constraints can be active or not, according to the state of the system. This particular structure results from the choice to simulate constraints

on the space position of points rigidly bound to the system. So, if this vector is of the form:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ y_1 \\ y_2 \\ \vdots \\ z_1 \\ z_2 \\ \vdots \end{pmatrix} \tag{2.2}$$

the unilateral constraints are directly $y \geq 0$, and the bilateral constraints are of the form $(x, z) = Constante$.

The **ContactVector** function returns a vector in this form, with (x_k, y_k, z_k) the position of the point of contact k. The points of contact are choosen among the tags. The tags choosen as contact points are specified in the vector `points_contact` defined in the **AdditionnalData.maple** file (see figure 2.5). The position of the contact points depend only of the position and the orientation of the different segments of the model, that is to say depends only of the state vector q. We will call "`Contact` vector" the vector returned by the **ContactVector** function.

The **ContactVector** function is used to generate the **Contact.c** file. After the linking of this function in Scilab, the user can call the **Contact** function as showed in the following example:

```
> C = Contact(q);
```

where q is the state vector and C is the vector of size $3 * NumberOfContacts$ defined in the equation 2.2.

Contact jacobian: the ContactJacobian.c file

The **ModelGeneration.maple** file contains the **ContactJacobianMatrix** function which returns the jacobian of the `Contact` vector defined in the equation 2.2. The returned value is a matrix which has the same structure as the `Contact` vector. It is composed of three parts of same size:

$$\begin{pmatrix} C_t^1 \\ C_n \\ C_t^2 \end{pmatrix} \tag{2.3}$$

where $C_t^1 = \partial\varphi_t^1/\partial q$ and $C_t^2 = \partial\varphi_t^2/\partial q$ are the jacobians of the bilateral constraints and $C_n = \partial\varphi_n/\partial q$ is the jacobian of the unilateral constraints. Because the `Contact` vector depends only of the state vectror q, jacobian depends only of q too.

The **ContactJacobianMatrix** function is used to generate the **ContactJacobian.c** file. After its linking in Scilab, the user can call the **ContactJacobian** function as shown in the following example:

```
> C = ContactJacobian(q)
```

where q is the state vector and C is the matrix of $3 * NumberOfTags$ rows and $NDOF$ columns ($NDOF$ is the number of degrees of freedom and is equal to the size of q) defined in the equation 2.3.

Contact hessian: the ContactHessian.c file

The **ModelGeneration.maple** file contains the **ContactHessian** procedure which computes the following vector the structure being the same than the `Contact` vector:

$$\begin{pmatrix} s_t^1(q,\dot{q}) \\ s_n(q,\dot{q}) \\ s_t^2(q,\dot{q}) \end{pmatrix} \tag{2.4}$$

with

$$s_t(q,\dot{q}) = \begin{pmatrix} s_t^1(q,\dot{q}) \\ s_t^2(q,\dot{q}) \end{pmatrix} \tag{2.5}$$

and $s_n(q,\dot{q})$ are terms appearing in the second differential of the bilateral and unilateral constraints respectively. The definition of these terms is made in section **??**.

The **ContactHessianMatrix** function is used to generate the **ContactHessian.c** file. After its linking in Scilab, the user can call the **ContactHessian** function as shown in the following example:

```
> s = ContactHessian(q, qdot)
```

where q and $qdot$ are the state vector and its differential and s is the vector of $3*NumberOfTags$ rows defined in the equation 2.4.

2.2 Dynamical Model

The dynamical model is meanly defined in the **DynamicData.maple** file. In this file, the inertials parameters (the gravity vector, the mass of the segment, the position of the segments

centers of mass and the inertia matrix of these segments relative to the centers of their attached frames) are defined. The figure 2.6 shows an example of this file.

The inertial parameters are explained in the section 2.2.1. Then, the computation of the inertia matrix and of the vector of non-linear effects is explained in sections 2.2.2 and 2.2.3.

```
# Vecteur gravité
Gravity := vector([0, -9.81, 0]):

# Solide 1 : (inexistant)
m_1  := 0:
G_1  := vector([0, 0, 0]*1e-3):
IO_1 := matrix([[0, 0, 0],[0, 0, 0],[0, 0, 0]]*1e-2):

# Solide 2 : (inexistant)
m_2  := 0:
G_2  := vector([0, 0, 0]*1e-3):
IO_2 := matrix([[0, 0, 0],[0, 0, 0],[0, 0, 0]]*1e-2):

# Solide 3 : (inexistant)
m_3  := 0:
G_3  := vector([0, 0, 0]*1e-3):
IO_3 := matrix([[0, 0, 0],[0, 0, 0],[0, 0, 0]]*1e-2):

# Solide 4 : (inexistant)
m_4  := 0:
G_4  := vector([0, 0, 0]*1e-3):
IO_4 := matrix([[0, 0, 0],[0, 0, 0],[0, 0, 0]]*1e-2):

# Solide 5 : (inexistant)
m_5  := 0:
G_5  := vector([0, 0, 0]*1e-3):
IO_5 := matrix([[0, 0, 0],[0, 0, 0],[0, 0, 0]]*1e-2):

# Solide 6 : pelvis
m_6  := 8.8:
G_6  := vector([0, 12, -68]*1e-3):
IO_6 := matrix([[13, 0, 0],[0, 14, 2],[0, 2, 5]]*1e-2):

# Solide 7 : equerre hanche droite
m_7  := 3.2:
G_7  := vector([5, -29, -107]*1e-3):
IO_7 := matrix([[6, 0, 0],[0, 5, -1],[0, -1, 1]]*1e-2):
.
.
.
.
.
.
# Solide 18 : cardan gauche
m_18  := 0.18:
G_18  := vector([0, 0, 0]*1e-3):
IO_18 := matrix([[0, 0, 0],[0, 0, 0],[0, 0, 0]]*1e-2):

# Solide 19 : pied gauche
m_19  := 2.34:
G_19  := vector([59, 0, 19]*1e-3):
IO_19 := matrix([[2, 0, 0],[0, 3, 0],[0, 0, 1]]*1e-2):

# Solide 20 : (inexistant)
m_20  := 0:
G_20  := vector([0, 0, 0]*1e-3):
IO_20 := matrix([[0, 0, 0],[0, 0, 0],[0, 0, 0]]*1e-2):

# Solide 21 : levier lombaires
m_21  := 1.05:
G_21  := vector([0, 47, -25]*1e-3):
IO_21 := matrix([[0, 0, 0],[0, 0, 0],[0, 0, 1]]*1e-2):

# Solide 22 : cardan armoire
m_22  := 0.46:
G_22  := vector([0, 0, 46]*1e-3):
IO_22 := matrix([[1, 0, 0],[0, 0, 0],[0, 0, 1]]*1e-2):

# Solide 23 : support armoire + armoire
m_23  := 48:
G_23  := vector([405, 13, 9]*1e-3):
IO_23 := matrix([[126, 6, -6],[6, 1075, 0],[-6, 0, 1026]]*1e-2):
```

Figure 2.6: The **DynamicData.maple** file describes different inertial parameters of the model.

2.2.1 Definition of the inertial parameters

The **DynamicData.maple** file defines the following inertial parameters:

- the vector `Gravity` gives the gravity in the reference frame (see figure 2.6);
- the segment masses `m_i` where i is the number of the considered segment (see figure 2.6);
- the positions `G_i` (i is the segment number) of the segment centers of mass in the frame attached to these segments (see figure 2.6);
- the inertia matrix relative to the center of the frame attached to the segment, `IO_i` where i is the segment number (see figure 2.6).

2.2.2 Inertia matrix: the Inertia.c file

The **ModelGeneration.maple** file contains the **GenerateInertiaMatrix** procedure which uses the **InertiaRecursion** function to generate the **Inertia.c** file. After its linking in Scilab, the user can call the **Inertia** function as shown in the following example:

```
> M = Inertia(q);
```

where q is the state vector and M is the inertia matrix defined in the equation 2.1. This inertia matrix is a square matrix of size $NDOF \times NDOF$ where $NDOF$ is the number of degrees of freedom of the model.

2.2.3 Non-linear effects: the NLEffects.c file

The **ModelGeneration.maple** file contains the **GenerateNLEffectsVector** procedure which uses the **NLEffectsRecursion** function to generate the **NLEffects.c** file. After its linking in Scilab, the user can call the **NLEffects** function as shown in the following example:

```
> N1 = NLeffects(q, qdot);
```

where q and *qdot* are the state vector and its differential and $N1$ is the vector composed of the non-linear effects and of the effects of the gravity:

$$N1 = N(q, \dot{q})\dot{q} + G(q) \tag{2.6}$$

$N(q, \dot{q})$ and $G(q)$ are defined in the equation 2.1. The NLEffects vector $N1$ is a vector of size $NDOF$ which is the number of degrees of freedom of the model.

2.3 VRML animation computation

to be completed

Chapter 3

A Biomecanic Model of a Human : Human36

3.1 Kinematic Model Definition

3.1.1 Joints definition

The Human36 model has the following 36 joints drawn on figure 3.1. These 36 joints are:

- ankle flexion/extension and internal/external rotation: 2dof;
- knee flexion/extension: 1dof;
- hip flexion/extension, internal/external rotation and abduction/adduction: 3dof;
- thorax (joint center = vertebra T10) flexion/extension, internal/external rotation and abduction/adduction: 3dof;
- sterno-clavicular articulation: elevation/lowering and antepulsion/retropulsion : 2dof
- shoulder flexion/extension, internal/external rotation and abduction/adduction: 3dof;
- elbow flexion/extension, internal/external rotation: 2dof;
- wrist flexion/extension and abduction/adduction: 2dof;
- head (joint center = vertebra C7) flexion/extension, internal/external rotation and abduction/adduction: 3dof;

According to the figure 3.1, the position vector q is:

$$q = \begin{pmatrix} q_1 \\ \vdots \\ q_{36} \\ q_{37} \\ \vdots \\ q_{39} \\ q_{40} \\ \vdots \\ q_{42} \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ b \\ c \end{matrix}}\right\} \text{Articular coordinates} \\ \left.\vphantom{\begin{matrix} a \\ b \\ c \end{matrix}}\right\} \text{Global translation} \\ \left.\vphantom{\begin{matrix} a \\ b \\ c \end{matrix}}\right\} \text{Global rotation} \end{matrix}$$

The frame between the feet of the model is the reference frame. Its origin is the projection on the ground of the middle of the centers of the ankle joints when the model is in the zero-position. The **Human36** model is in the zero-position when the body is standing up, the feet flat on the ground spaced apart about the same distance as the width of the hips, the arms

are straight and parallel with the sides of the body and the palms of the hands are facing to the front. The x-axis of the reference frame is the direction to front of the model, the y-axis is vertically upward and the z-axis is to the right of the model. In zero-position, each frame attached to a segment is oriented as the reference frame and its origin is the joint center of this segment. For example, in the zero-position, the frame attached to the right shoulder is the frame which origin is on the right sternoclavicular joint and which is oriented as the reference frame.The origin of the different segments of the **Human36** model are given in table 3.1.

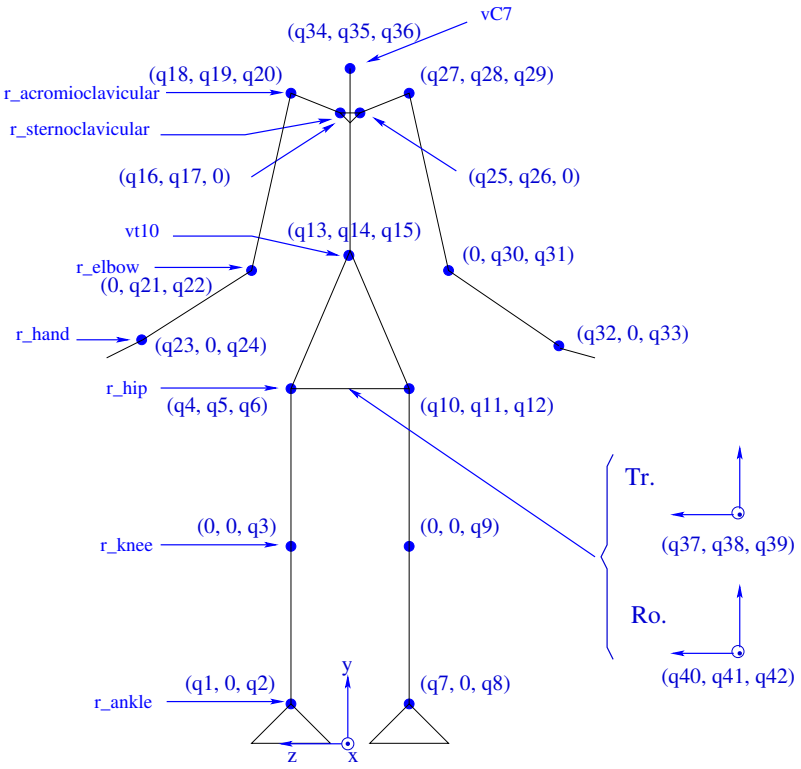


Figure 3.1: Articular notations of Human36. The notation is (rotation around x, rotation around y, rotation around z) or (translation along x, translation along y, translation along z) for the global position (cf Tr.) We followed the ISB recommendations [9, 10] for the components $q_{1\ldots36}$

3.1.2 Anatomic Model Lengths and zero-position

The figures 3.2 and 3.3 give the zero-position (the position for which q is the null vector) and the 31 anatomical lengths ($L(1\ldots31)$) necessary to the construction of the model. These lengths

Segment name	Origin of the attached frame
Right thigh	right hip
Right shank	right knee
Right foot	right ankle
Left thigh	left hip
Left shank	left knee
Left foot	left ankle
Upper Part of Trunk (Thorax)	vt10
Right shoulder (Clavicule/Scapula)	right sternoclavicular
Right upper-arm	right shoulder joint
Right forearm	right elbow
Right hand	right wrist
Left shoulder (Clavicule/Scapula)	left sternoclavicular
Left upper-arm	left shoulder joint
Left forearm	left elbow
Left hand	left wrist
Head (and neck)	vc7

Table 3.1: Name of the segments and origin of the frame attached to these segments

correspond to the positions of the center of the joints relatively to an adjacent one and to other lengths used to compute the inertia characteristics of the extremities (hand, feet and head).

These lengths are not fixed on the model. The user can modify them by giving :

- either the values of all lengths (that is to say by defining the $L(1 \dots 31)$ vector);
- or the value of the size of the model. The lengths will be computed from anthropometric data which are explained below.

The way to set these lengths is explained in the section 3.1.2.

Anthropometric Data

Most of the choosen anthropometric data have been founded in the De Leva paper [6], but other anthropometric sources have been used. All the choices are explained below. Lengths relative to the size of the body are given in percentage. The implemented model is a male.

De Leva [6] has adjusted the Zatsiorsky data in order to consider as references the centers of joints or other commonly anatomical landmarks rather than the rarely used Zatsiorsky

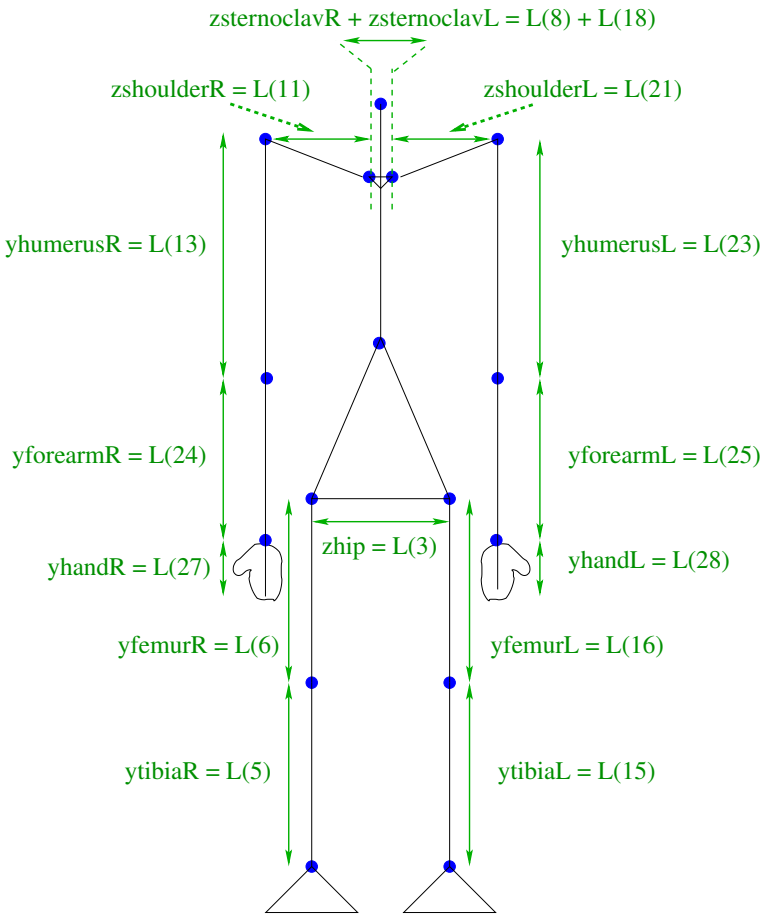


Figure 3.2: Zero Position (front view) and lengths notations of Human36

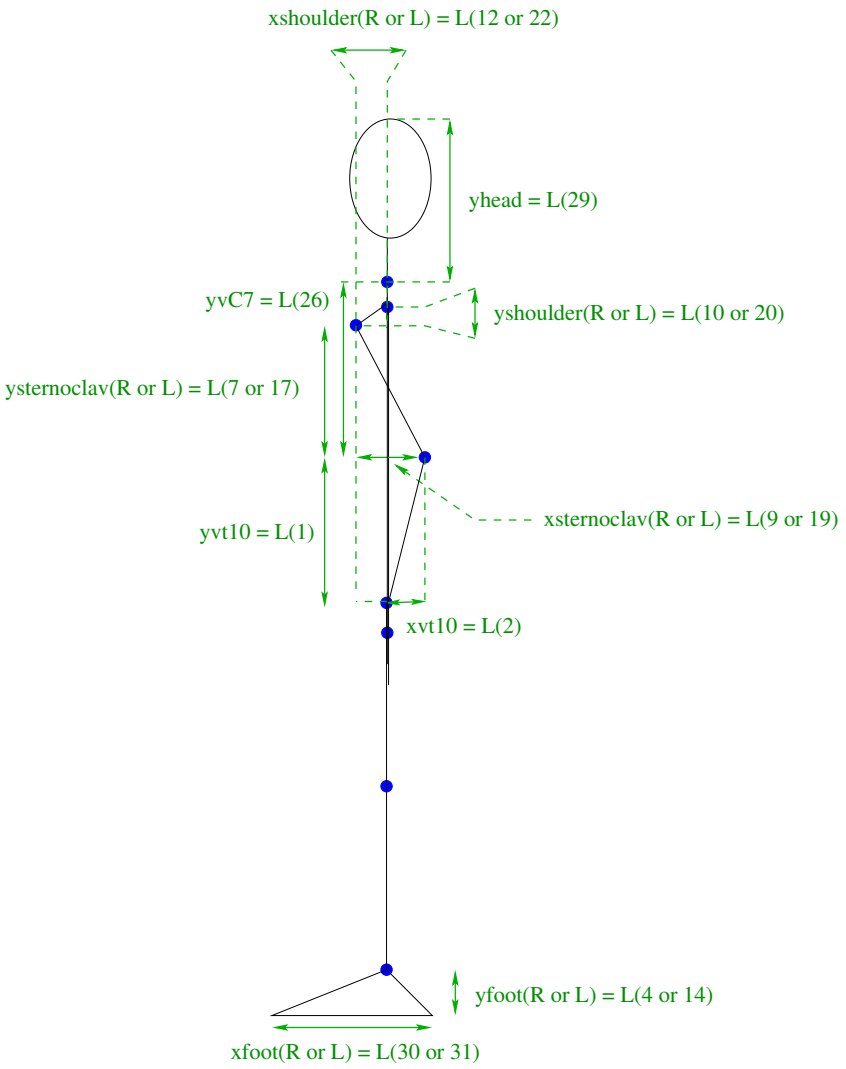


Figure 3.3: Zero Position (side view) and lengths notations of Human36

original landmarks. De Leva [6] gives some mean longitudinal lengths for females(mean mass = 61.9kg and mean stature = 1.735m) and males (mean mass = 73.0kg and mean stature = 1.741m). But he gives only longitudinal lengths. Other anthropometric data sources have been used to compute the non-longitudinal lengths.

Mean longitudinal lengths (along the y-axis) :

Some longitudinal lengths are directly given by De Leva in [6]. They are given in millimeters. We have divided these lengths by the mean stature (1.741 m) to obtain longitudinal lengths relative to the model size in percent.

The following longitudinal lengths have been computed from De Leva [6]:

- **Lower part of trunk**:
  This lower part is delimited by the horizontal plane containing the centers of the hip joints ($plane_1$) and by the one containing the Xyphoid process and the T10 vertebra ($plane_2$). We assume that the Xyphoid process is in the same horizontal plane as the T10 vertebra (see figure 3.4). Then we can compute the mean length between these two planes by computing the sum of the length between $plane_1$ and the horizontal plane containing the omphalion (center of the navel) and of the length between this last plane and $plane_2$. Then, we have:

$$\begin{aligned} yvt10 = L(1) &= \frac{length(LPT) + length(MPT^*)}{mean\ stature}\ Size \\ &= \frac{145.7 + 215.5}{1741}\ Size \\ &= 0.2075\ Size \end{aligned}$$

  with the De Leva [6] notations (LPT and MPT are the De Leva [6] Lower and Middle Parts of Trunk).

- **Upper part of trunk** :
  This upper part is delimited by the horizontal plane containing the Xyphoid process and the T10 vertebra and the one containing the suprasternale. This mean measure is given by De Leva [6].

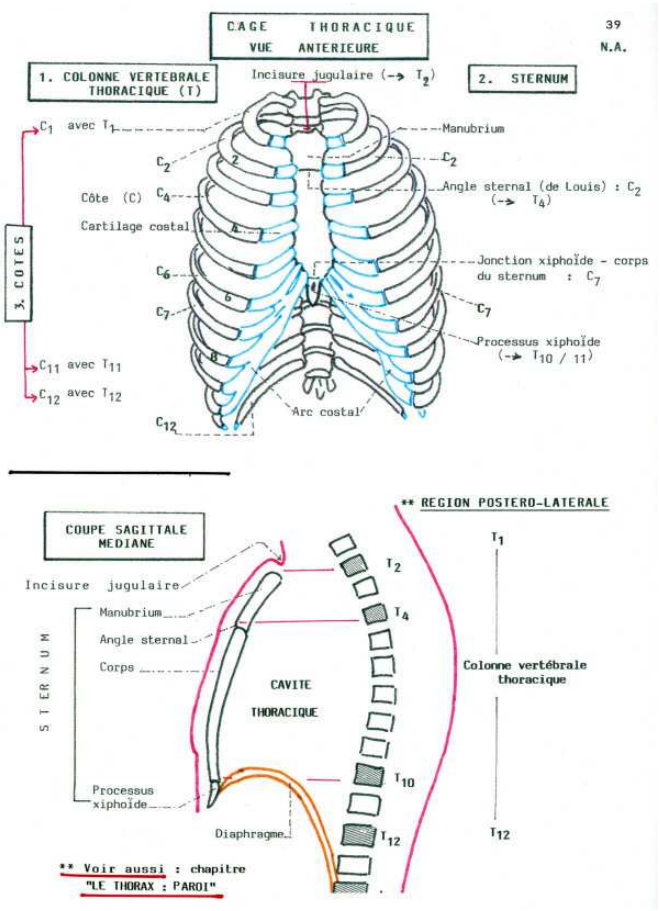


Figure 3.4: The Xyphoid process and the T10 vertebra can be considered in the same horizontal plane. Scheme realized by the Pr. NGUYEN HUU (Brest Medecine Faculty) [7].

- **Longitudinal length between the T10 and the C7 vertebra** :
  This mean length corresponds to the distance between the horizontal plane containing the Xyphoid process and the T10 vertebra and the one containing the C7 vertebra. This length is given by Zatsiorsky *et al.*

- **Vertical length between the centers of sterno-clavicular joint and the shoulder joint** :
  De Leva [6] gives the Zatsiorsky mean longitudinal lengths between the trochanterion and the suprasternale and between the trochanterion and the acromion. The longitudinal

distance between the suprasternale and the acromion is:

$$\begin{aligned} yshoulder(right\ or\ left) &= L(10\ or\ 20) \\ &= \frac{(Troch.\text{-}acromion) - (Troch.\text{-}suprasternale)}{mean\ stature}\ Size \\ &= \frac{(553.2 - 535.1)}{1741}\ Size \\ &= 0.0104\ Size \end{aligned}$$

- **Foot Heigth** :
  De Leva gives the following distances:
  - `VERT/CERV`: longitudinal distance between the Vertex and the Cervicale;
  - `CERV/MIDH`: longitudinal distance between the Cervicale and the hip joint center middle point;
  - `HJC/KJC`: longitudinal distance between the hip joint center and the knee joint center;
  - `KJC/LMAL`: longitudinal distance between the knee joint center and the lateral malleolus.

  Then, we have:

$$\begin{aligned} yfoot &= L(4\ or\ 14) \\ &= (1 - \frac{\text{VERT/CERV} + \text{CERV/MIDH} + \text{HJC/KJC} + \text{KJC/LMAL}}{mean\ stature})\ Size \\ &= (1 - \frac{242.9 + 603.3 + 422.2 + 434.0}{1741})\ Size \\ &= 0.022\ Size \end{aligned}$$

The mean longitudinal lengths are shown in table 3.2.

Mean transversal lengths (along the z-axis) :

In paper [6], De Leva combines different sources of anthropometric data, and uses particulary the Clauser *et al* [4]. That is why we tried to find most lengths in Clauser studies. Other

Longitudinal length specification	Length Notation in Human36	Length (% of Size)	Bibliographic origin
Lateral malleolus/Knee Joint Center	L(5)/L(15)	24.93	De Leva
Knee Joint Center/Hip Joint Center	L(6)/L(16)	24.25	De Leva
Hip Joint Center/T10 vertebra	L(1)	20.75	De Leva
T10 vertebra/Suprasternale	L(7)/L(17)	9.80	De Leva
Suprasternale/Shoulder Joint Center	L(10)/L(20)	1.04	De Leva
Shoulder Joint Center/Elbow Joint Center	L(13)/L(23)	16.18	De Leva
Elbow Joint Center/Wrist Joint Center	L(24)/L(25)	15.44	De Leva
T10 vertebra/C7 vertebra	L(26)	13.9	De Leva
Foot heigth	L(4)/L(14)	2.22	De Leva
Wrist Joint Center/3rd Dactilion	L(27)/L(28)	10.91	De Leva
C7 vertebra/Vertex	L(29)	13.95	De Leva

Table 3.2: Longitudinal lengths between center of joints.

lengths were found in Dempster studies [2].

The three following lengths are required:

- **horizontal length between the centers of hip joints** :
  The mean distance between the center of hip joints was computed from a report by Clauser *et al* [4], based on the analysis of 13 male cadavers(mean age = 49 ans, mean stature = 1.7272m). Clauser *et al* [4] measured the mean data `HIP BREADTH` which is the mean "horizontal distance across the greatest lateral protusion of the hips". We assume that the centers of hip joints are on the centers of the thighs (along the z-axis). Then we assume that the distance between the centers of the two hip joints is equal to the horizontal length at the upper part of the thigh which is equal to the `HIP BREADTH` mean value. Then the distance between the centers of the hip is set to the half of the `HIP BREADTH` mean value. We divided this result by the mean stature in order to have it depending on the model size. Then, we have:

$$zhip = \frac{HIP\ BREADTH}{2 * ESTIMATED\ STATURE}\ Size = 0.1002\ Size$$

- **horizontal length between the centers of the two sternoclavicular joints** :

  This length was found in the Dempster studies [2]. Dempster measured a distance of one

inch between the sternoclavicular joints. Then, we have:

$$zsternoclav(right\ or\ left) = L(8\ or\ 18) = \frac{(one\ inch)}{2}$$

- **horizontal length between the centers of the sternoclavicular joint and the shoulder joint** :

  This length was found in the Dempster studies [2]. Dempster set the horizontal distance between the centers of the two shoulder joints to $0.259 * Size$. Then, we have:

$$zshoulder(right\ or\ left) = L(11\ or\ 21) = \frac{0.259\ Size}{2} - zsternoclav(right\ or\ left)$$

Mean depth lengths (along the x-axis) :

The following depth lengths were computed:

- **depth length between the middle point of the center of the two hip joints and the T10 vertebra** :

  We assume that the depth distance between the centers of the hip joints middle point and the T10 vertebra is the Clauser *et al* [4] `WAIST DEPTH/OMPH` half length. This length is the vertical distance between the measuring table and the anterior surface of the body at the level of omphalion. Then, we have:

$$xvt10 = L(2) = \frac{WAISTDEPTH/OMPH}{2} = 90.85 \text{ mm}$$

- **depth length between the T10 vertebra and the center of the sternoclavicular joint** :

  We assume that the T10 vertebra and the center of the sternoclavicular joint are at the same distance as the vertical plane containing the center of the hip joints and the y-axis of the reference frame. Then, the depth length between the T10 vertebra and the centers of the two sternoclavicular joints are :

$$xsternoclav(right\ or\ left) = L(9\ or\ 19) = WAIST\ DEPTH/OMPH = 181.7 \text{ mm}$$

- **depth length between the centers of the sternoclavicular and the shoulder joint** :

  We assume that the center of the shoulder jointxs lay on the vertical plane containing the center of the hip joint and the y-axis of the reference frame (see [3, page 25]). Then, the depth lengths between the centers of the sternoclavicular joints and the shoulder joints are:

$$xshoulder(right\ or\ left) = L(12\ or\ 22) = \frac{WAIST\ DEPTH/OMPH}{2} = 90.85 \text{ mm}$$

Anatomical lengths getting and setting

Anatomical lengths setting:

The user can modify the model anatomical lengths $L(1 \dots 31)$ by two ways:

- If the user knows only the size of the subject, he can set it in the model and the anatomical lengths will be computed from the anthropometric data explained in the section 3.1.2. In this case, the user set the model size by calling the **SetModelSize** function. The size is given in meter. The following example shows the use of this function:

```
> ModelSize = 1.80;
> SetModelSize(ModelSize);
```

- The user can set straightly the different anatomical lengths by calling the **SetAnatomicalLengths** function. The lengths are given in meter. The following example shows the use of this function:

```
> AnatLengths = .....
>....
> SetAnatomicalLengths(AnatLengths);
```

Anatomical lengths getting :

There are default anatomical lengths in the model. These default lengths have been computed with the preceding anthropometric data and a size of 1.741m.

The user can get these lengths from the model by calling the **GetAnatomicalLengths** function. This function returns the $L(1 \dots 31)$ row vector. The following example shows how to use it:

```
> AnatLengths = GetAnatomicalLengths();
```

The user can also get the size used in the model by calling the **GetModelSize** function:

```
>ModelSize = GetModelSize();
```

3.2 Tags Model

The figures 3.5 and 3.6 show the anatomic landmarks corresponding to the tags. The number and the name of the tags are specified in the table 3.3. The model construction needs the position (x, y, z) of the tags in their attached segment frame (see section 3.1.1 for the definition of these segment frames). We have 28 tags, then the model construction needs 84 lengths. The ERGODATA measures dictionnary [5] describes most of these anatomical landmarks.

For a given position q, we can compute the position of the tags in the reference frame by calling the **Tags** function. This function returns a matrix T of $(NumberOfTags+1)$ rows and 3 columns. The row i ($i \leq NumberOfTags$) of this matrix is the position (x, y, z) of the tag i in the reference frame. The last row (the $(NumberOfTags+1)$ row) is the position of the model global center of mass. Then, T is on the form:

$$T = \begin{pmatrix} x^0_{tag_1} & y^0_{tag_1} & z^0_{tag_1} \\ x^0_{tag_2} & y^0_{tag_2} & z^0_{tag_2} \\ \vdots & \vdots & \vdots \\ x^0_{tag_{NumberOfTags}} & y^0_{tag_{NumberOfTags}} & z^0_{tag_{NumberOfTags}} \\ x^0_{CenterOfMass} & y^0_{CenterOfMass} & z^0_{CenterOfMass} \end{pmatrix}$$

where the 0 represents the reference frame. The following example shows the use of the **Tags** function:

```
> exec Load.sci;
> q = zeros(33,1);
> T = Tags(q);
```

3.2.1 Anthropometric data

The model needs the position (x, y, z) of the tags in their attached segment frame. Then, it needs $3 * NumberOfTags$ lengths.

Anthropometric data from different sources give us these lengths relativly to the subject height $Size$. We used mainly three sources:

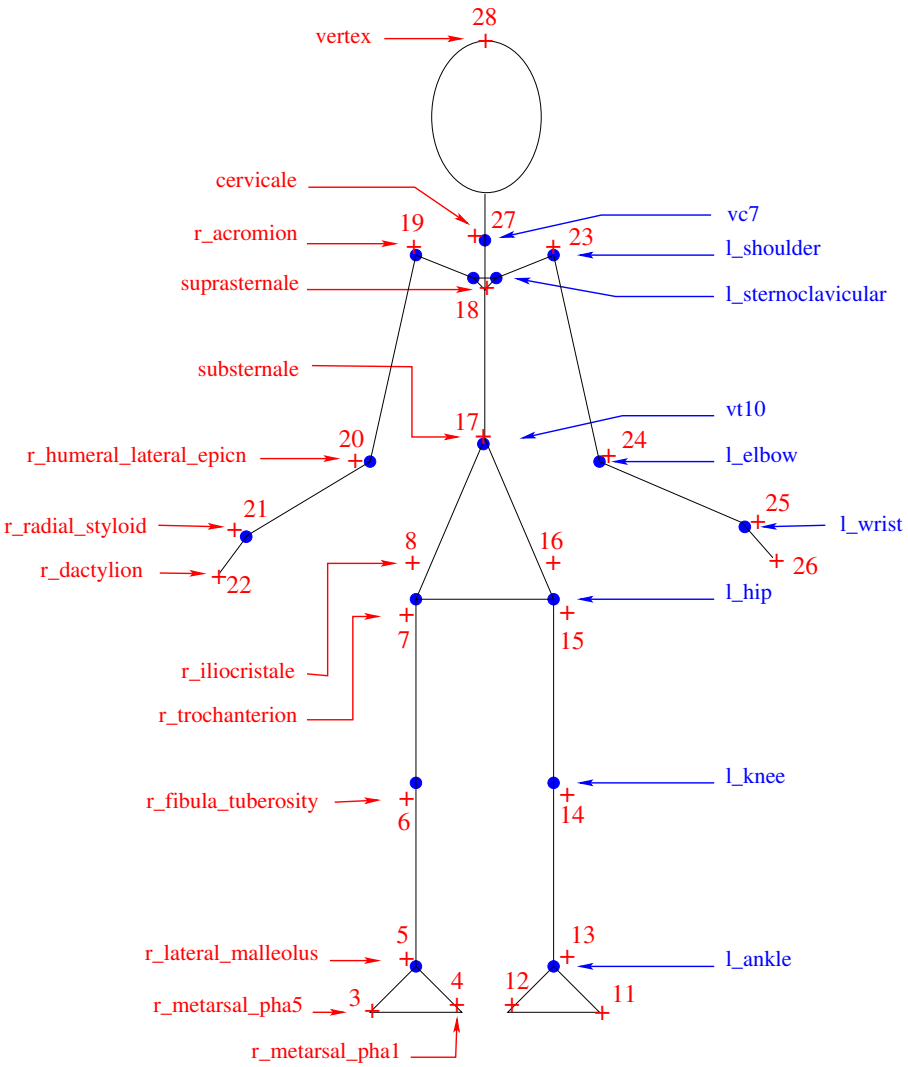


Figure 3.5: Front view of Human36. Tags names, tags numbers and names of articulations.

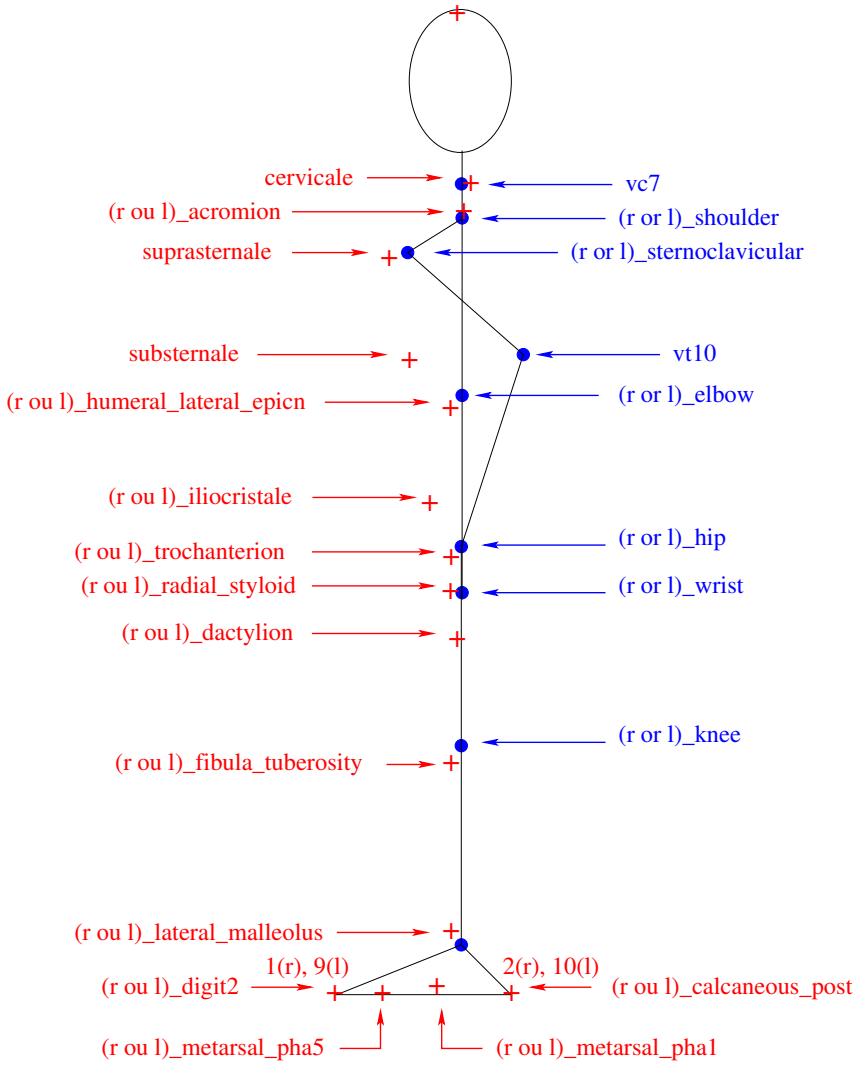


Figure 3.6: Profile view of Human36

Anatomic names	H-anim tags names	tags number	attached segment
Right second digit	r_digit2	1	right foot
Right Calcaneous post	r_calcaneous_post	2	right foot
Right fifth metatarsal	r_metatarsal_pha5	3	right foot
Right first metatarsal	r_metatarsal_pha1	4	right foot
Right lateral malleolus	r_lateral_malleolus	5	right shank
Right femoral lateral epicondyle	r_femoral_lateral_epicn	6	right thigh
Right great trochanterion	r_trochanterion	7	right tigh
Right iliocristale	r_iliocristale	8	lower part of trunk
Left second digit	l_digit2	9	left foot
Left Calcaneous post	l_calcaneous_post	10	left foot
Left fifth metatarsal	l_metatarsal_pha5	11	left foot
Left first metatarsal	l_metatarsal_pha1	12	left foot
Left lateral malleolus	l_lateral_malleolus	13	left shank
Left femoral lateral epicondyle	l_femoral_lateral_epicn	14	left thigh
Left great trochanterion	l_trochanterion	15	left thigh
Left iliocristale	l_iliocristale	16	lower part of trunk
Xyphoid process	substernale	17	upper part of trunk
Suprasternale	suprasternale	18	upper part of trunk
Right acromion	r_acromion	19	right shoulder
Right humeral lateral epicondyle	r_humeral_lateral_epicn	20	right arm
Right radial styloid	r_radial_styloid	21	right forearm
Right third dactylion	r_dactylion	22	right hand
Left acromion	l_acromion	23	left shoulder
Left humeral lateral epicondyle	l_humeral_lateral_epicn	24	left arm
Left radial styloid	l_radial_styloid	25	left forearm
Left third dactylion	l_dactylion	26	left hand
Cervicale	cervicale	27	upper part of trunk
Vertex	vertex *	28	head

Table 3.3: Correspondence between anatomic names, H-anim names [1], tags numbers and the segments the tags are attached to

- the De Leva one [6];
- the Clauser *et al* [4] one. In this case the `ESTIMATED STATURE` mean measure is used. It is the mean stature of the Clauser sample subjects;
- An other anthropometric data source are obtained by experiments for motion capture (see the `Tools/Reconstruction` directory). In this experiments, measurements of some

anatomical landmarks positions were made. The set of the measurements used to compute our anatomical landmarks positions are given in the table 3.4.

Subject	1	2	3	4	5	6	7	8	9	10	11
age	20	26	24	25	23	24	25	24	25	26	23
height	170	182	170	183	172	187	193	180	170	180	178
x digit 2	20.5	23	24	?	25	?	23.5	20.5	?	20.5	19
x pha5	12.5	10	12	14.5	10	12.5	11.5	15	11.5	10.5	10
z right pha5	4.5	6	5	6	5	6	7	5	5.5	5	5
z right lateral malleolus	4.5	4.5	4.5	6.5	3.5	5.5	4	4.5	4	3.5	4

Table 3.4: Anatomical landmarks positions measured in motion capture experiments. These positions are given in the frame attached to these landmarks. The age is in years and the lengths in cm.

Second digit position

The second digits attached segment is the foot.

We assume that the second digit lays on the ground plane (and then its y component is the height of the foot) and on the frontal plane containing the center of the corresponding ankle joint (and then its z component is null). The measurements of the experiments (see table 3.4) allow us to compute the mean second digit position relative to the height of the subject. The mean measure of the second digit position x component is 22 and the mean stature corresponding is 178.125. Then the position of the right or left second digit in respectively the right or left foot frame is:

$$\begin{aligned}(x_{DIGIT2}, y_{DIGIT2}, z_{DIGIT2}) &= (\frac{\text{mean } x \text{ measure}}{\text{mean stature}}\ Size, -L(4\ or\ 14), 0) \\ &= (\frac{22}{178.125}\ Size, -L(4\ or\ 14), 0) \\ &= (0.1235\ Size, -L(4\ or\ 14), 0)\end{aligned}$$

Calcaneous post position

The Calcaneous post attached segment is the feet.

As for the second digit landmark, we assume that the calcaneous post landmark lays on the ground plane (and then its y component coorespond to the height of the foot) and on the

frontal plane containing the center of the corresponding ankle joint (and then its z component is null).

De Leva [6] gives us the mean length of the foot from the heel to the tip of the longest toe. Then the x component of this landmark is:

$$\begin{aligned} x_{CALC} &= -(\frac{\text{De Leva mean length foot}}{\text{De Leva mean stature}}\ Size - x_{DIGIT2}) \\ &= -(\frac{258.1}{1741}\ Size - 0.1235\ Size) \\ &= -0.0248\ Size \end{aligned}$$

Then the right or left calcaneous post positions in the respective rigth or left foot frame are:

$$(x_{CALC}, y_{CALC}, z_{CALC}) = (-0.0248\ Size, -L(4\ or\ 14), 0)$$

Fifth metatarsal position

The fifth metatarsal attached segment is the foot. We assume (even if it is wrong for a real human) that the fifth metatarsal landmark lays on the ground. Then its y component coresponds to the height of the foot. The x and z components of the fifth metatarsal position were computed from measurements of motions capture experiments. For the right fifth metatarsal the computed mean measures are:

- x mean measure = 11.82cm and the corresponding mean stature is equal to 178.64cm;
- z mean measure = 5.45cm and the corresponding mean stature is equal to 178.64cm;

Then the right fifth metatarsal position in the right foot frame is:

$$\begin{aligned} (x_{PHA5}, y_{PHA5}, z_{PHA5}) &= (\frac{11.82}{178.64}\ Size, -L(4), \frac{5.45}{178.64}\ Size) \\ &= (0.0662\ Size, -L(4), 0.0305\ Size) \end{aligned}$$

and the left one in the left foot frame is:

$$(x_{PHA5}, y_{PHA5}, z_{PHA5}) = (0.0662\ Size, -L(14), -0.0305\ Size)$$

First metatarsal position

The first metatarsal attached segment is the foot. We assume (even if it is wrong for a real human) that the first metatarsal landmark lays on the ground. Then its y component coresponds to the height of the foot. We assume for simplicity that the first metatarsal landmark is symmetrical to the fifth by the sagittal plane containing the center of the corresponding ankle joint. Then the right and left first metatarsal positions in the respective right and left foot frames are given by:

$$\begin{aligned}(x_{PHA1}, y_{PHA1}, z_{PHA1}) &= (x_{PHA5}, y_{PHA5}, -z_{PHA5}) \\ &= (0.0662\ Size, -L(4), -0.0305\ Size) \quad \text{(right)} \\ &= (0.0662\ Size, -L(14), 0.0305\ Size) \quad \text{(left)}\end{aligned}$$

Lateral malleolus position

The lateral malleolus attached segment is the shank. We assume that the lateral malleolus landmarks lay on the transversal and the frontal plane containing their cooresponding (right or left) ankle joint center. Then their x component is null and their y component corresponds to the length of the shanks.

The z components were computed from measurements of motion capture experiments (see table 3.4). The mean measure of this z component is 4.45cm and the corresponding mean stature is 178.64cm. Then the right lateral malleolus position in the right shank frame is:

$$\begin{aligned}(x_{MALL}, y_{MALL}, z_{MALL}) &= (0, -L(5), \frac{\text{experiences mean } z \text{ measure}}{\text{experiences mean stature}}\ Size) \\ &= (0, -L(5), 0.0249\ Size)\end{aligned}$$

and the left one in the left shank frame is:

$$(x_{MALL}, y_{MALL}, z_{MALL}) = (0, -L(15), -0.0249\ Size)$$

Femoral lateral epicondyle position

The femoral lateral epicondyle attached segment is the thigh. We assume that this anatomical landmark lays on the transversal and the frontal planes containing the center of the knee joint. Then the x component of this landmark is null and the y one is the length between the centers of the hip joint and the knee joint.

Clauser *et al* [4] gives the mean `KNEE BREADTH(BONE)` measurement corresponding to the mean "maximum distance between the right femoral epicondyles by exerting sufficient pressure to compress the tissue overlying the femur". Then the position of the right femoral lateral epicondyle (FEM) in the right thigh frame is:

$$\begin{aligned}(x_{FEM}, y_{FEM}, z_{FEM}) &= (0, -L(6), \frac{\text{KNEE BREADTH(BONE)}}{2 * \text{(ESTIMATED STATURE)}}\ Size) \\ &= (0, -L(6), \frac{10.01}{172.72}\ Size) \\ &= (0, -L(6), 0.0290\ Size)\end{aligned}$$

and the position of the left femoral lateral epicondyle in the left thigh frame is:

$$(x_{FEM}, y_{FEM}, z_{FEM}) = (0, -L(16), -0.290\ Size)$$

Great trochanterion position

The right and left great trochanters attached segment are the the right and left thigh respectively. We assume that the great trochanters lay on the transversal and the frontal planes containing the centers of the hip joints. Then the x and y component of their position in the lower part of trunk frame are null.

Clauser *et al*[4] gives the mean `Bitrochanteric Breadth(Bone)` measurement corresponding to the mean "horizontal distance between the maximum protusion of the right and left greater trochantor exerting sufficient pressure to compress the tissue overlying the femurs". Then the z component of the right great trochanter position is:

$$\begin{aligned}(x_{TROCH}, y_{TROCH}, z_{TROCH}) &= (0, 0, \frac{\text{Bitrochanteric Breadth(Bone)}}{2 * \text{(ESTIMATED STATURE)}}\ Size - \frac{L(3)}{2}) \\ &= (0, 0, \frac{32.51}{2 * 172.72}\ Size - \frac{L(3)}{2}) \\ &= (0, 0, 0.0941\ Size - \frac{L(3)}{2})\end{aligned}$$

and those of the left great trochanter position is:

$$(x_{TROCH}, y_{TROCH}, z_{TROCH}) = (0, 0, -0.0941\ Size + \frac{L(3)}{2})$$

Iliac crest position

The iliac crest landmarks attached segment is the lower part of trunk.

Clauser *et al* [4] gives the `BI-SPINOUS BREADTH` measurement which is the mean "horizontal distance between the right and left anterior-superior iliac spines". We have then, for the right iliac crest:

$$\begin{aligned} z_{ILIA} &= \frac{\text{BI-SPINOUS BREADTH}}{2 * (\text{ESTIMATED STATURE})} \, Size \\ &= \frac{24.08}{2 * 172.72} \, Size \\ &= 0.0697 \, Size \end{aligned}$$

De Leva [6] gives the mean vertical distance between the center of the hip joint and the iliospinale for the men (63.7mm).

The x component of the iliospinale is computed from measurement of movement reconstruction data (see table 3.4). We use the mean of the measure for each subject. We have then:

$$\begin{aligned} \text{mean measure} &= 4.83\text{cm} \\ \text{mean stature} &= 178.33\text{cm} \\ x_{ILIA} &= \frac{\text{mean measure}}{\text{mean stature}} \, Size \\ &= 0.0271 \, Size \end{aligned}$$

The right and left iliac crest positions in the lower part of trunk frame are:

$$(x_{ILIA}, y_{ILIA}, z_{ILIA})(\text{right or left}) = (0.0271 \, Size, 0.0366 \, Size, (+ \; or \; -)0.0697 \, Size)$$

Xyphoid process position

The Xyphoid process attached segment is the thorax. We assume that the y and z components of the position of the Xyphoid process in the thorax frame are null (see figure 3.4).

Clauser *et al* [4] gives the `CHEST DEPTH` measurement corresponding to the "vertical distance between the measuring table and the anterior surface of the body at the level of the thelion".

The thelion is the center of the nipples. The Xyphoid x component is approximated by this measure. Then the Xyphoid process position in the thorax frame is:

$$\begin{aligned}(x_{XYP}, y_{XYP}, z_{XYP}) &= (\frac{\text{CHEST DEPTH}}{\text{ESTIMATED STATURE}}, 0, 0) \\ &= (\frac{21.06}{172.72}\ Size, 0, 0) \\ &= (0.1219\ Size, 0, 0)\end{aligned}$$

Suprasternale position

The suprasternale landmark attached segment is the thorax. It lays between the center of the right and left sternoclavicular joints. Then we have:

$$(x_{SUPR}, y_{SUPR}, z_{SUPR}) = (\frac{L(9) + L(19)}{2}, \frac{L(7) + L(17)}{2}, 0)$$

Acromion position

The acromion landmark attached segment is the shoulder. De Leva [6] gives the mean longitudinal length from acromion to the center of the corresponding shoulder joint. We assume that the acromion is at the verticale of the center of the corresponding shoulder joint. The figure 3.7 shows the position of the acromions in their attached frame (which are the corresponding sternoclavicular frames).

Then we have for the right acromion:

$$\begin{aligned}(x_{ACR}, y_{ACR}, z_{ACR}) &= (-L(12), L(10) + \frac{\text{De Leva mean length(acromion-shoulderJCs)}}{\text{mean stature}}\ Size, L(11)) \\ &= (-L(12), L(10) + \frac{34.5}{1741}\ Size, L(11)) \\ &= (-L(12), L(10) + 0.0198\ Size, L(11))\end{aligned}$$

and for the left one:

$$(x_{ACR}, y_{ACR}, z_{ACR}) = (-L(22), L(20) + 0.0198\ Size, -L(21))$$

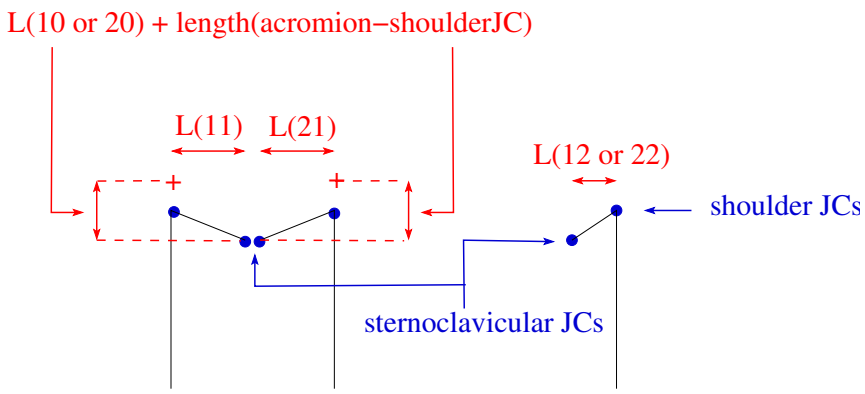


Figure 3.7: Position of the acromion landmarks relative to the center of sternoclavicular joint (JC corresponds to Joint Center).

Humeral lateral epicondyle position

The humeral lateral epicondyle attached segment is the corresponding upper-arm. We assume that this landmark lays on the transversal and on the frontal (palm upwards) planes containing the center of the elbow joint. Then the x component of its position in the upper-arm frame is null. And the y one corresponds to the upper-arm length.

Clauser *et al* [4] gives the `ELBOW BREADTH/BONE` measurement corresponding to the mean "maximum dixtance between the humeral epicondyles exerting sufficient pressure to compress the tissue overlying the humerus". Then the position of the right humeral lateral epicondyle in the right arm frame is:

$$\begin{aligned}(x_{HUM}, y_{HUM}, z_{HUM}) &= (0, -L(13), \frac{\text{ELBOW BREADTH/BONE}}{2 * (\text{ESTIMATED STATURE})}\ Size) \\ &= (0, -L(13), \frac{7.27}{2 * 172.72}\ Size) \\ &= (0, -L(13), 0.0211\ Size)\end{aligned}$$

and the position of the left humeral lateral epicondyle in the left arm frame is:

$$(x_{HUM}, y_{HUM}, z_{HUM}) = (0, -L(23), -0.0211\ Size)$$

Styloid position

The styloid landmark attached segment is the forearm. De Leva [6] gives the mean longitudinal length from the elbow joint center (EJC) and the normal projection on the forearm longitudinal axis of the Styloid. Then, for the right and left styloid, the y component is:

$$\begin{aligned} y_{STYL} &= -\frac{\text{EJC-StyloidProjection}}{\text{mean stature}}\ Size \\ &= -\frac{266.9}{1741}\ Size \\ &= -0.1533\ Size \end{aligned}$$

We assume that the right and left styloid lay on the frontal plane. Then, their x component are null. Clauser *et al* [4] gives the `WRIST BREADTH/BONE` measurement corresponding to the "maximum distance between the radical and ulnar styloid process exerting sufficient pressure to compress the tissue overlying the radius and ulna". Then we have for the right styloid:

$$\begin{aligned} z_{STYL} &= \frac{\text{WRIST BREADTH/BONE}}{\text{ESTIMATED STATURE}} Size \\ &= \frac{5.72}{172.72}\ Size \\ &= 0.0331\ Size \end{aligned}$$

The right and left styloid position are:

$$(x_{STYL}, y_{STYL}, z_{STYL})(\text{right or left}) = (0, -0.1533\ Size, (+\ or\ -)0.0331\ Size)$$

Third dactylion position

The third dactylion attached segment is the hand. We assume that the x and z components are null. De Leva [6] gives the mean longitudinal length from the 3rd dactylion (`DAC3`) to the center of the Wrist Joint (`WJC`). Then, we have for the right and left third dactylion:

$$\begin{aligned} (x_{DAC3}, y_{DAC3}, zDAC3) &= (0, -\frac{\text{WJC-DAC3}}{\text{mean stature}}\ Size, 0) \\ &= (0, -\frac{189.9}{1741}\ Size, 0) \\ &= (0, -0.1091\ Size, 0) \end{aligned}$$

Cervicale position

The cervicale attached segment is the thorax. The Cervicale landmark lays on the back and on the base of the neck (at the same height than the C7 vertebra). Then the z component of the position is null and the y one is the vertical length between the T10 and the C7 vertebra, that is to say the length $L(26)$. We assume that in the sagittal plane the cervicale landmark is between the C7 and the T10 vertebra.

Clauser *et al* [4] gives the `Neck Depth` measurement. This measure is the maximum depth of the neck perpendicular to the long axis of the neck. Then the x component of the cervicale position is:

$$\begin{aligned} x_{Cerv.} &= L(2) - \frac{\text{Neck Depth}}{2 * \text{ESTIMATED STATURE}}\ Size \\ &= L(2) - 0.0392\ Size \end{aligned}$$

The Cervicale position in the thorax frame is:

$$(x_{Cerv.}, y_{Cerv.}, z_{Cerv.}) = (L(2) - 0.0392\ Size, L(26), 0)$$

Vertex position

The vertex tag lays on the y-axis of its attached segment frame. De Leva [6] gives the longitudinal length of the head (from cervicale to vertex). This length was divided by the mean stature in order to have the length, in percentage, relative to the model size. Then the position of the tag in its attached segment frame is:

$$(x_{Vertex}, y_{Vertex}, z_{Vertex}) = (0, 0.1395\ Size, 0)$$

Landmarks positions recapitulated table

The table 3.5 recapitulates the position of the tags in their attached frames.

3.2.2 Tags to Joint centers lengths setting and getting

The preceding section gives the anatomical landmarks position in their attached segment frame. These positions are relative to the model height and were computed from anthropometric sources. By default, the tags' positions are computed as in the preceding section with a model size of 1.741m (this size is the De Leva [6] mean stature data). The user can modify these positions by setting them in the model as explained in the following section.

Tag name	**Tag x component**	**Tag y component**	**Tag z component**
Right second digit	0.1235*Size	-L(4)	0
Right Calcaneous post	-0.0248*Size	-L(4)	0
Right fifth metatarsal	0.0662*Size	-L(4)	0.0305*Size
Right first metatarsal	0.0662*Size	-L(4)	-0.0305*Size
Right lateral malleolus	0	-L(5)	0.0249*Size
Right femoral lateral epicondyle	0	-L(6)	0.0290*Size
Right great trochanterion	0	0	0.0941*Size - $\frac{L(3)}{2}$
Right iliocristale	0.0271*Size	0.0366*Size	0.0697*Size
Left second digit	0.1235*Size	-L(14)	0
Left Calcaneous post	-0.0248*Size	-L(14)	0
Left fifth metatarsal	0.0662*Size	-L(14)	-0.0305*Size
Left first metatarsal	0.0662*Size	-L(14)	0.0305*Size
Left lateral malleolus	0	-L(15)	-0.0249*Size
Left femoral lateral epicondyle	0	-L(16)	-0.0290*Size
Left great trochanterion	0	0	-0.0941*Size + $\frac{L(3)}{2}$
Left iliocristale	0.0271*Size	0.0366*Size	-0.0697*Size
Xyphoid process	0.1219*Size	0	0
Suprasternale	$\frac{L(9)+L(19)}{2}$	$\frac{L(7)+L(17)}{2}$	0
Right acromion	-L(12)	L(10) + 0.0198*Size	L(11)
Right humeral lateral epicondyle	0	-L(13)	0.0211*Size
Right radial styloid	0	-0.1533*Size	0.0331*Size
Right third dactylion	0	-0.1091*Size	0
Left acromion	-L(22)	L(20) + 0.0198*Size	-L(21)
Left humeral lateral epicondyle	0	-L(23)	-0.0211*Size
Left radial styloid	0	-0.1533*Size	-0.0331*Size
Left third dactylion	0	-0.1091*Size	0
Cervicale	L(2) - 0.0392*Size	L(26)	0
Vertex	0	0.1395*Size	0

Table 3.5: Anatomical landmarks positions in their attached segment frame

Tags to Joint centers lengths setting

The user can modify the tags positions by:

- setting the size of the model. In this case, the new tags positions will be computed as in the preceding section with the size set by the user. The function allowing to set the model size is the **SetModelSize** function described in section 3.1.2;

- setting all the positions by calling the **SetTag2JointLengths** function. This function's input is a row vector of size $3 * NumberOfTags$ of the following form:

$$Tag2JointLengths = (\underbrace{x_1, y_1, z_1}_{\text{tag 1}}, \underbrace{x_2, y_2, z_2}_{\text{tag 2}}, \ldots, \underbrace{x_{nb_{tags}}, y_{nb_{tags}}, z_{nb_{tags}}}_{\text{tag } NumberOfTags})$$

  where the number of the tags are specified in section 3.2. The following example shows the use of this function:

```
> Tag2JointLengths = []...
> SetTag2JointLengths(Tag2JointLengths);
```

Tags to Joint centers lengths getting

The user can get the tags positions in their attached segment frames by calling the **GetTag2JointLengths** function. This function returns the $Tag2JointLengths$ row vector defined in the preceding section. The following example shows how to use it:

```
> Tag2JointLengths = GetTag2JointLengths();
```

3.3 Dynamical Model

The segments masses and the position of the centers of mass relative to joint position of the segment are found from anthropometrical data

The segments numerotation in **HuMAnS** is given in the table 3.6.

3.3.1 Segment Masses

De Leva [6] gives the segment masses relative to the body mass. The table 4.2 gives the proportions. The shoulders are not take into account in the dynamical model. Then their masses are set to 0.

Segments masses and global mass setting

The user can modify the body mass with the **SetMass** function. The mass must be given in kg. This function set the user input mass in the model. The following example shows the use of this function:

Segment name	Segment number
Lower Part of Trunk	1
Right thigh	2
Right shank	3
Right foot	4
Left thigh	5
Left shank	6
Left foot	7
Upper Part of Trunk (Thorax)	8
Right shoulder (Clavicule/Scapula)	9
Right upper-arm	10
Right forearm	11
Right hand	12
Left shoulder (Clavicule/Scapula)	13
Left upper-arm	14
Left forearm	15
Left hand	16
Head (and neck)	17

Table 3.6: Segment numerotation in **HuMAnS**

```
> subjectMass = 63;
> SetMass(subjectMass);
```

When the body mass is modified with this function, the segment masses are modified as specified in the preceding section.

Segments masses and global mass getting

The user can get the body mass by using the **GetMass** function. The default body mass is 73kg (the mean male mass found by De Leva [6]). The following example shows the use of this function:

```
> subjectMass = GetMass();
```

Segment name	Mass (%)
Lower Part of Trunk	27.5
Right thigh	14.16
Right shank	4.33
Right foot	1.37
Left thigh	14.16
Left shank	4.33
Left foot	1.37
Upper Part of Trunk (Thorax)	15.96
Right shoulder (Clavicule/Scapula)	0
Right upper-arm	2.71
Right forearm	1.62
Right hand	0.61
Left shoulder (Clavicule/Scapula)	0
Left upper-arm	2.71
Left forearm	1.62
Left hand	0.61
Head (and neck)	6.94

Table 3.7: Segment masses relative to the body mass

3.3.2 Segments centers of mass

We assume that the segment center of mass lay on the respective segment longitudinal axis. De Leva [6] gives the segments center of mass longitudinal position relative to the respective segment longitudinal length. In De Leva [6], segments center of mass longitudinal position is referenced either to proximal or cranial points. The table 3.8 gives the relative longitudinal position of the center of mass.

In order to compute the longitudinal Center of mass position of our lower part of trunk (between hip joints center and Xyphoid process/T10 vertebra), we use :

$$mO_1G = m_{LPT}O_1G_{LPT} + m_{MPT}O_1G_{MPT}$$

and

$$m_{Trunk}O_1G_{Trunk} = m_{LPT}O_1G_{LPT} + m_{MPT^*}O_1G_{MPT^*} + m_{UPT}O_1G_{UPT}$$

where m, m_{LPT}, m_{MPT^*}, m_{UPT} and m_{Trunk}, G, G_{LPT}, G_{MPT^*}, G_{UPT} and G_{Trunk} are respectively the mass and the centers of mass of our lower part of trunk, of the De Leva [6] Lower Part of Trunk, of the Zatsiorsky Middle Part of Trunk and of the De Leva [6] Upper Part of

Trunk and O_1 the center of our lower part of trunk frame (which is the hip joint centers middle point).

Segment name	**Origin Point**	**End Point**	**Longitudinal COM Position** (in %)	**Bibliographic Origin**
Head	cervicale	vertex	49.98	Zatsiorsky et al
Upper Part of Trunk	xyphoid process	suprasternale	70.01	De Leva [6]
Lower Part of Trunk	hip JCs middle point	xyphoid process	51.08	De Leva [6]
Thigh	hip JC	knee JC	40.95	De Leva [6]
Shank	knee JC	Lateral malleolus	44.59	De Leva [6]
Foot	heel	acropodion	44.15	De Leva [6]
Upper arm	shoulder JC	elbow JC	57.72	De Leva [6]
Forearm	elbow JC	wrist JC	45.74	De Leva [6]
Hand	wrist JC	3rd dactilion	36.91	Zatsiorsky et al

Table 3.8: Longitudinal relative position of the segment center of mass. The acropodion is the tip of the longest toe(first or second)

The y-axis of the lower part of trunk, thighs, shanks, upper-arms, forearms, hands and head segments are the longitudinal axis of these segments. Then, the x and z components of the position of these centers of masse are null.

We model the foot with a cylinder of diameter the foot height. Then the longitudinal axis of the foot is parallel with the x-axis of the foot frame and contained in the xOy plane and at a distance of half the foot height from the x-axis. Then the positions of the centers of mass of the feet in the feet segment frames are:

$$(x_G, y_G, z_G) = (0.4415\ L(30) + x_{\text{right heel}},\ \frac{-L(4)}{2},\ 0) \quad \text{(right foot)}$$

$$(x_G, y_G, z_G) = (0.4415\ L(31) + x_{\text{left heel}},\ \frac{-L(14)}{2},\ 0) \quad \text{(left foot)}$$

where $L(30)$, $L(31)$, $L(4)$ and$L(14)$ are respectively the lengths of the right and left foot and the heights of the right and left foot (see section 3.1.2) and $x_{\text{right heel}}$ and $x_{\text{left heel}}$ the x component of the position of the right and left heels in the right and left foot frame.

We model the upper part of trunk with a cylinder of diameter the distance l between the T10 vertebra and the projection of the suprasternale on the xOz plane. Then this longitudinal axis is parallel with the y-axis of the upper part of trunk frame and distant to the y-axis of half this distance l. Then the position of the upper part of trunk center of mass in its attached

frame is:

$$(x_G, y_G, z_G) = (\frac{L(9) + L(19)}{4},\ 0.7001\ L(7),\ 0)$$

where $L(9)$, $L(19)$ and $L(7)$ are defined in section 3.1.2.

3.3.3 Inertia computation

We want to know the inertia matrix of each segment relative to the segment frame center. We assume that the y-axis of the segments frames are their longitudinal segments and that it is an axis of symetry of the segment. Then the non-diagonal components of these matrices are null. Then we use the radii of gyration given by de Leva [6] and the Huygens theorem to compute the diagonal components (in fact, the sagittal, transversal and longitudinal axes relative to which the radii of gyration are given are the axes containing the segment center of mass G and parallel respectively with the segment frame x, z and y axis). For example, the moment of inertia about the x-axis of the segment frame is:

$$I_{O\vec{x}} = I_{G\vec{x}} + md^2$$

where O, G, O_x, G_x are respectively the segment frame center, the segment center of mass, the axis $O\vec{x}$ and $G\vec{x}$, d is the perpendicular distance between the $O\vec{x}$ and $G\vec{x}$ axis and m the segment mass. Furthermore, we have:

$$I_{G\vec{x}} = mR^2_{G\vec{x}}$$

with R the radius of gyration relative to the $G\vec{x}$-axis.

The x-axis can be replaced by the y and z axes. Finally, we have:

$$I_O = \begin{pmatrix} I_{O\vec{x}} & 0 & 0 \\ 0 & I_{O\vec{y}} & 0 \\ 0 & 0 & I_{O\vec{z}} \end{pmatrix}$$

The segment mass is computed as in the section 4.3.1 and the length d is computed from the data contained in the preceding section. The radii of gyration $R_{G(\vec{x},\vec{y} \text{ or } \vec{z})}$ relative to their segment longitudinal lengths are given in the table 3.9. The radii of gyration of our lower part of trunk was not given by De Leva [6]. The following section shows how these radii of gyration were computed.

Computation of our lower part of trunk raddi of gyration

Our lower part of trunk is composed of the De Leva [6] lower part of trunk (LPT) and the Zatsiorsky middle part of trunk (MPT^*). Then we can compute the moment of inertia $I_{G\vec{x}}$ on

Segment name	$R_{G\vec{x}}$ (in %)	$R_{G\vec{y}}$ (in %)	$R_{G\vec{z}}$ (in %)
Lower part of trunk	27.22	26.28	22.6
Right thigh	32.9	14.9	32.9
Right shank	25.5	10.3	24.9
Rigth foot	12.4	25.7	24.5
Left thigh	32.9	14.9	32.9
Left Shank	25.5	10.3	24.9
Left foot	12.4	25.7	24.5
Upper part of trunk	71.6	65.9	45.4
Right shoulder	0	0	0
Right upper arm	28.5	15.8	26.9
Right forearm	27.6	12.1	26.5
Right hand	28.8	18.4	23.5
Left shoulder	0	0	0
Left arm	28.5	15.8	26.9
Left forearm	27.6	12.1	26.5
Left hand	28.8	18.4	23.5
Head	30.3	26.1	31.5

Table 3.9: Radii of gyration $R_{G(\vec{x},\vec{y} \text{ or } \vec{z})}$ relative to their segment longitudinal lengths (G is the corresponding segment center of mass). We assume that the shoulder masses are null then the radii of gyration are set to 0.

the $G\vec{x}$-axis of our lower part of trunk with:

$$\begin{aligned} I_{G\vec{x}} &= I_{LPT,\ G\vec{x}} + I_{MPT^*,\ G\vec{x}} \\ &= m_{LPT} R^2_{LPT,\ G\vec{x}} + m_{MPT^*} R^2_{MPT^*,\ G\vec{x}} \end{aligned}$$

where G is the center of mass of our lower part of trunk, $I_{LPT,\ G\vec{x}}$ and $I_{MPT^*,\ G\vec{x}}$ are respectively the moment of inertia of the De Leva [6] lower part of trunk and of the Zatsiorsky middle part of trunk on the $G\vec{x}$ axis. The computation with the corresponding raddi of gyration is valid because we assume that the center of mass of our lower part of trunk, of the De Leva [6] lower part of trunk and of the Zatsiorsky middle part of trunk lay on the $O\vec{x}$ axis where O is the origin of our lower part of trunk (*i.e* the hip joint center middle point).

Then, we can compute the lower part of trunk radius of gyration $r_{G\vec{x}}$ relative to its mean longitudinal length $\overline{l}$ with:

$$r_{G\vec{x}} = \frac{1}{\overline{l}} \sqrt{\frac{I_{G\vec{x}}}{m_{LPT} + m_{MPT^*}}}$$

The x-axis can be replaced with the y or z axis.

Chapter 4

Model of a biped robot : Kondo KHR-1

4.1 Kinematic Model Definition

4.1.1 Joints definition and zero-position

The joints of the Kondo model are shown on the figure 4.1. The Kondo model has the following 17 joints:

- ankle flexion/extension and internal/external rotation: 2dof ((q2,q1) and (q7,q6));
- knee flexion/extension: 1dof (q3 and q8);
- hip flexion/extension and internal/external rotation : 2dof ((q4,q5) and (q9,q10));
- shoulder flexion/extension and internal/external rotation : 2dof ((q12,q11) and (q15,q14));
- wrist flexion/extension : 1dof (q13 and q16);
- head abduction/adduction: 1dof (q17);

Then, the position vector q is:

$$q = \begin{pmatrix} q_1 \\ \vdots \\ q_{17} \\ q_{18} \\ \vdots \\ q_{20} \\ q_{21} \\ \vdots \\ q_{23} \end{pmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{Articular coordinates} \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{Global translation} \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \text{Global rotation} \end{matrix}$$

The frame between the feet of the model is the reference frame. Its origin is the projection on the ground of the middle of centers of the ankle joints when the model is in the zero-position (the position for which q is the null vector, see figure 4.1). Its x-axis is the direction to front of the model, its y-axis is vertically upward and its z-axis is to the right of the model. In zero-position, the frames attached to the segments are all oriented as the reference frame and their origins are the center of the joint of this segment.

The direction of the rotation is normed, the rotation in the frontal plan is positive when the solid moves away from the sagittal plan, and the rotation in the sagittal plan is positive when the solid goes in the negative x-axis (see the figure 4.2). For the head, the norm chosen is that the rotation is positive in the trigonometrical sense in top view.

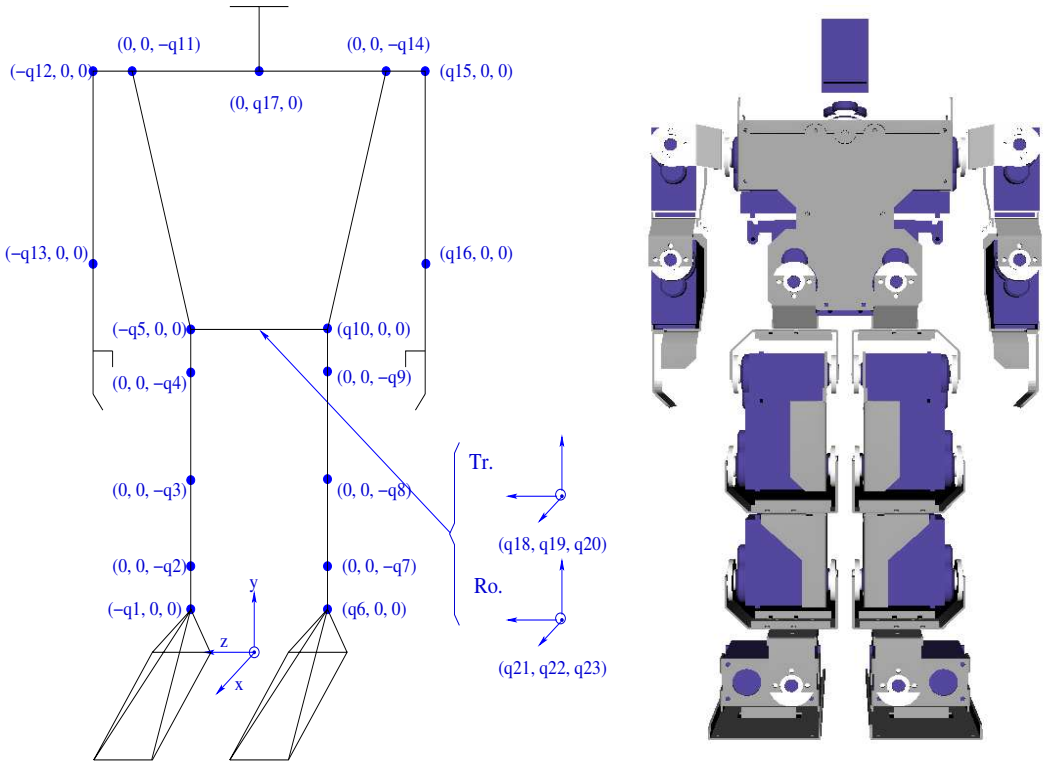


Figure 4.1: Articular notations of Kondo on the left. The notation is (rotation around x, rotation around y, rotation around z) or (translation along x, translation along y, translation along z) for the global position (cf Tr.). Zero position in VRML on the right.

4.1.2 Model Lengths

The figure 4.3 gives the 24 mechanical lengths ($l(1 \ldots 24)$) necessary to the construction of the model. These lengths define the positions of the centers of joint relative to its parent joint and other lengths used to compute the inertia characteristics of the extremities (hand, foot and head).

4.2 Tags Model

The figure 4.4 shows the mechanical landmarks corresponding to the tags. The name of the 34 tags are specified in the table 4.1. The model construction needs the position (x, y, z) of the tags in their attached segment frame (see section 4.1.1 for the definition of these segments frames).

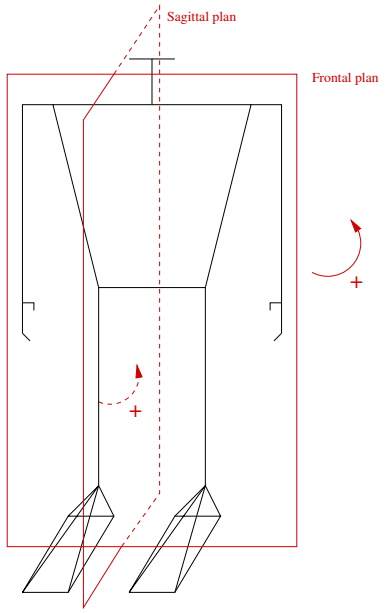


Figure 4.2: The norm of rotation sense

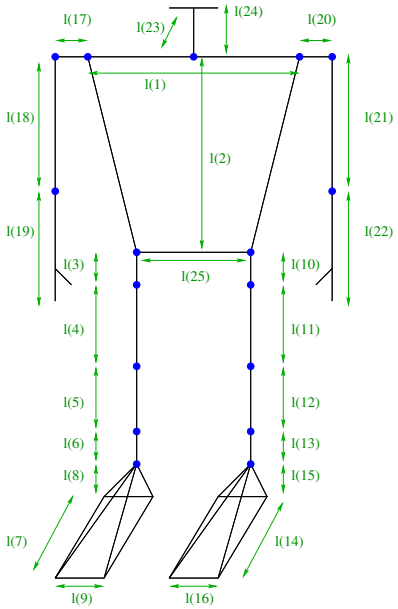


Figure 4.3: Zero Position and lengths notations of Kondo

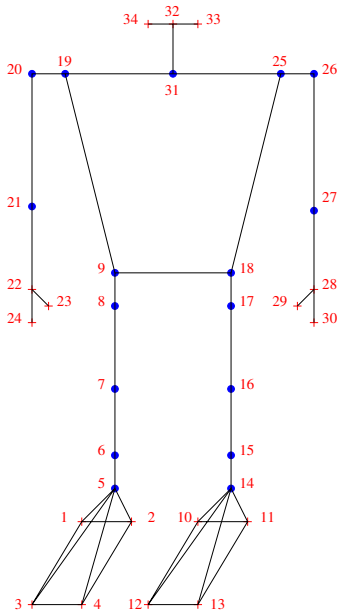


Figure 4.4: Inclined view of Kondo with tags numbers.

For a given position q, we can compute the tags positions in the reference frame by calling the **Tags** function. This function returns a matrix T of $(NumberOfTags + 1)$ rows and 3 columns. The row i ($i \leq NumberOfTags$) of this matrix is the position (x, y, z) of the tag i in the reference frame. The last row (the $(NumberOfTags + 1)$ row) is the position of the global center of mass of the model. Then, T is of the form:

$$T = \begin{pmatrix} x^0_{tag_1} & y^0_{tag_1} & z^0_{tag_1} \\ x^0_{tag_2} & y^0_{tag_2} & z^0_{tag_2} \\ \vdots & \vdots & \vdots \\ x^0_{tag_{NumberOfTags}} & y^0_{tag_{NumberOfTags}} & z^0_{tag_{NumberOfTags}} \\ x^0_{CenterOfMass} & y^0_{CenterOfMass} & z^0_{CenterOfMass} \end{pmatrix}$$

where the 0 represents the reference frame. The following example shows the use of the **Tags** function:

```
> exec Load.sci;
> q = zeros(23,1);
> T = Tags(q);
```

H-anim names	**Tags names**	**Tags number**
none	right foot heel right	1
none	right foot heel left	2
none	right foot toe right	3
none	right foot toe left	4
Kondo_r_ankle	right foot ankle on foot	5
Kondo_r_ankleSagittal	right foot ankle on leg	6
Kondo_r_knee	right knee	7
Kondo_r_hip	right great trochanter	8
Kondo_r_hipFrontal	right iliac crest	9
none	left foot heel right	10
none	Left foot heel left	11
none	Left foot toe right	12
none	Left foot toe left	13
Kondo_l_ankle	Left foot ankle on foot	14
Kondo_l_ankleSagittal	Left foot ankle on leg	15
Kondo_l_knee	Left knee	16
Kondo_l_hip	Left great trochanter	17
Kondo_l_hipFrontal	Left iliac crest	18
Kondo_r_shoulderSagittal	right clavicular	19
Kondo_r_shoulder	right shoulder	20
Kondo_r_elbow	Right arm bone	21
none	right fist	22
none	right thumb tip	23
none	right middle finger tip	24
Kondo_l_shoulderSagittal	Left clavicular	25
Kondo_l_shoulder	Left shoulder	26
Kondo_l_elbow	Left arm bone	27
none	Left fist	28
none	left thumb tip	29
none	left middle finger tip	30
Kondo_vc7	neck	31
none	nose	32
none	left ear	33
none	right ear	34

Table 4.1: Correspondence H-anim [1] names / model names and associated tags

4.3 Dynamical Model

The Kondo weights 1.3 kilogrammes. The segments masses and the position of the centers of mass (a vector relative to the solid reference) are found from experiment (this experiment is explained in section 4.3.2).

4.3.1 Segment Masses

The mass of each solid is deduced from the mass of the motor and aluminium pieces (obtained by weighing), the total robot mass (obtained by weighing too), and approximation of the wires (80% of the wires mass is located on the trunk, and 10% is located on each thigh).

Solid name	**Mass (%)**
Trunk	40.38
Right hip	0.77
Right thigh	7.69
Right leg	3.85
Right ankle	0.77
Right foot	5.38
Left hip	0.77
Left thigh	7.69
Left leg	3.85
Left ankle	0.77
Left foot	5.38
Right shoulder	0.77
Right arm	3.85
Right hand	5.00
Left shoulder	0.77
Left arm	3.85
Left hand	5.00
Head	3.46

Table 4.2: Segment masses relative to the body mass

4.3.2 Segments centers of mass

We model each solid as a parallelepipedon, in fact we reduce the solid to a motor. The center of mass is given by one experiment on a motor. In fact, this experiment is very brief. We put

the motor on a thin ruler, in the 2 basic axis in 2 plan, and we search the equilibrium position. It gives the 2 plans of equilibrium, and the intersection gives the position of center of mass. The result is given on figure 4.5.

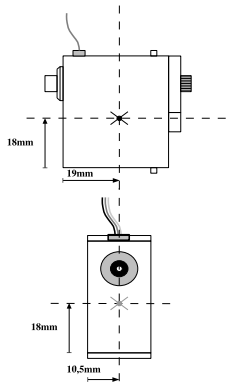


Figure 4.5: Position of mass center for motor.

For the solids more complicated as trunk and thigh , we make an approximation with the different motors present in it (2 for the thigh and 4 for the trunk, the battery is considered on center).

4.3.3 Inertia computation

We want to know the inertia matrix of each solid. As we said before, the solids are considered as parallelepipedon. The inertia matrix is known for this kind of geometry. See the figure 4.6.

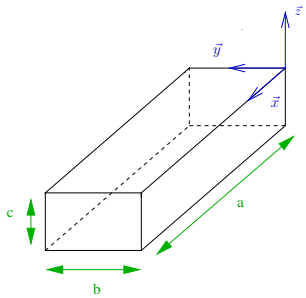


Figure 4.6: Inertia matrix for parallelepipedon.

Segment	Position of mass center		
	$\vec{x}$	$\vec{y}$	$\vec{z}$
Trunk	0	0.5	0
Right hip	0	-0.5	0
Right thigh	0	-0.5	0
Right leg	0	-0.6	0
Right ankle	0	-0.5	0
Right foot	0	0	0.25
Left hip	0	-0.5	0
Left thigh	0	-0.5	0
Left leg	0	-0.6	0
Left ankle	0	-0.5	0
Left foot	0	0	-0.25
Right shoulder	0	0	-0.5
Right arm	0	-0.4	0
Right hand	0	-0.5	0
Left shoulder	0	0	0.5
Left arm	0	-0.4	0
Left hand	0	-0.5	0
Head	0.4	0.6	0

Table 4.3: Different body mass center position in ratio of solid lengths, from the solid reference

Finally, we have:

$$I_G = \begin{pmatrix} I_{G\vec{x}} & 0 & 0 \\ 0 & I_{G\vec{y}} & 0 \\ 0 & 0 & I_{G\vec{z}} \end{pmatrix}$$

where $I_{G\vec{x}}$, $I_{G\vec{y}}$ and $I_{G\vec{z}}$ are given by :

$$\begin{cases} I_{G\vec{x}} &= \dfrac{m \times (b^2 + c^2)}{12} \\ I_{G\vec{y}} &= \dfrac{m \times (a^2 + c^2)}{12} \\ I_{G\vec{z}} &= \dfrac{m \times (a^2 + b^2)}{12} \end{cases}$$

4.3.4 VRML representation

A 3D visualization is available in VRML. It needs on input the articulations values and the vrml geometry file and creates a realistic animation. The names of solids and articulations are showen on figure 4.7.

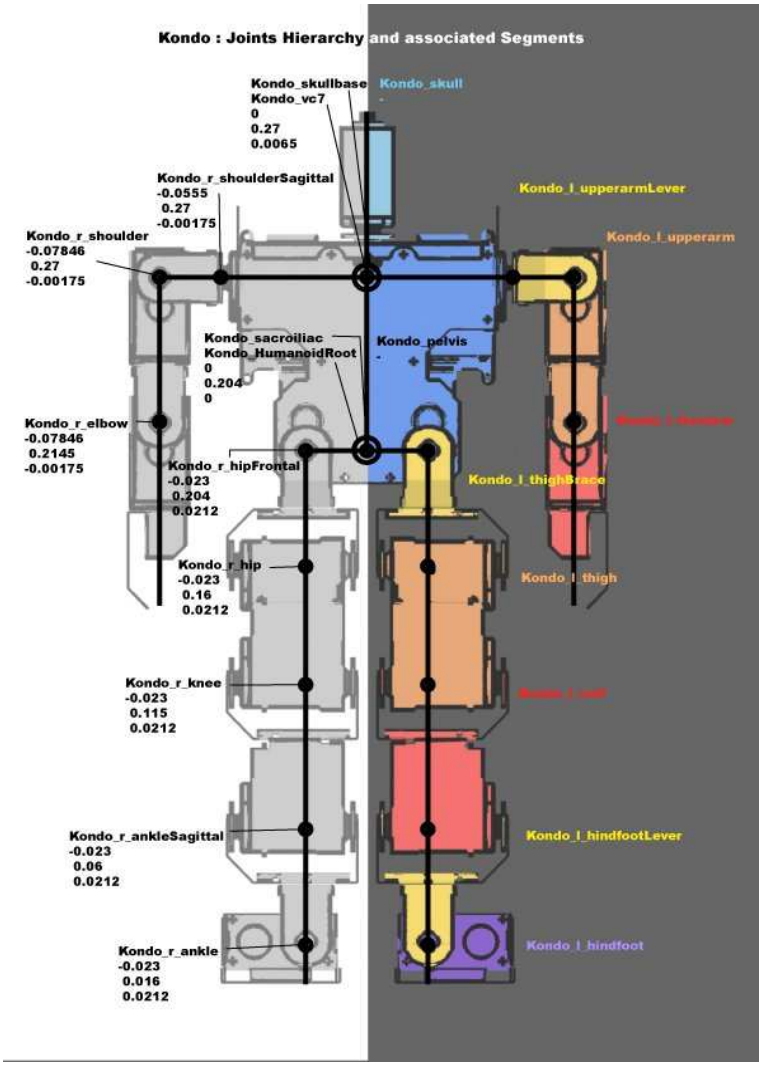


Figure 4.7: VRML model, in zero position, with names and positions of joints

Chapter 5

Simulation

5.1 Introduction

HuMAnS is a simulator of the "Event Driven" type. The dynamic system is handled as an ordinary differential equations broken by evenments as impacts or unstickings of the constraints, actuation events...

The ordinary differential equation integration with handling of evenments is implemented in the `simulation` function which is the main procedure of HuMAnS is located in the `Kernel` directory. This `simulation` function needs that the user specifies the initial and final time of the simulation, the sampling period and the initial state of the system. Then it returns the times of the sampling, the positions, the velocities and the corresponding states (z, Actuation and Contact).

So the `simulation` call is of the following form:

```
[T, Q, QDOT, Z, ACTUATIONSTATE, CONTACTSTATE] = ...
Simulation(InitialTime,FinalTime,SamplingPeriod,InitialPosition,InitialVelocity)
```

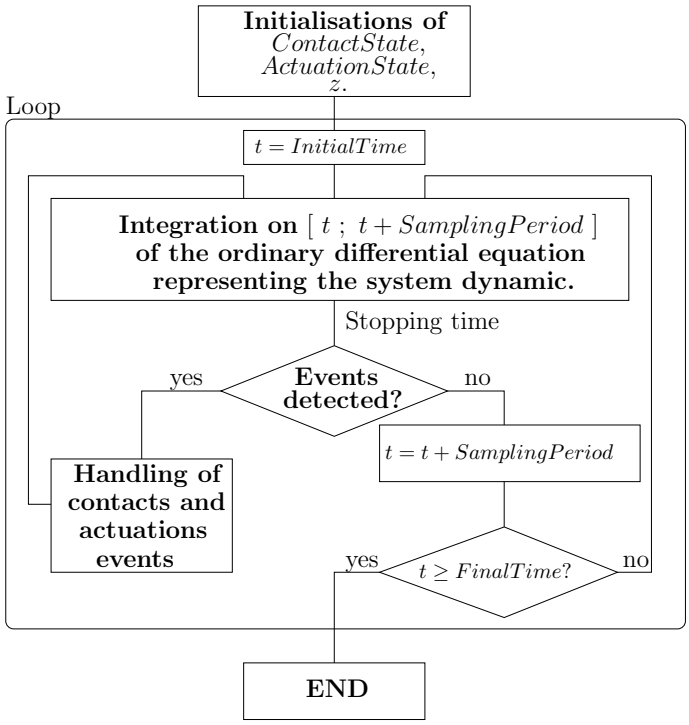


Figure 5.1: Simulation procedure scheme

The figure 5.1 describes the `simulation` function. After an initialization of the different state variables, it makes a loop in which it integrates between two sampling times the differential

equation corresponding to the dynamic system. If events occur, the integration stops, the simulator handles these events and restarts the integration at the stopping time. If no events occurs, the simulator restarts the integration at the next sampling time. And so on until the final time is reached.

5.2 Description of the ordinary differential equation corresponding to the dynamic system

We recall that the dynamic system can be considered as an ordinary differential equation succession broken by events. Then, the dynamic system can be written in the form:

$$\dot{x} = f(t, x)$$

where t is the time and x the state vector:

$$x = \begin{pmatrix} q \\ \dot{q} \\ z \end{pmatrix}$$

5.2.1 The CompleteDynamics function

In the simulator, the function f is the `CompleteDynamics` function. The call to this function is:

```
[xdot] = CompleteDynamics(t, x)
```

were x, $xdot$ and t are respectively the system state vector, the system state vector differential and the time.

The `CompleteDynamics` function computes the system state vector differential in two steps:

- first the `ActuationDynamics` function computes the z differential $\dot{z}$ and the Torques;
- then, the `LagrangianDynamics` function computes the acceleration $\ddot{q}$ from these Torques and contact state.

So the `CompleteDynamics` function returns the system state vector differential:

$$\dot{x} = \begin{pmatrix} \dot{q} \\ \ddot{q} \\ \dot{z} \end{pmatrix} = \text{CompleteDynamics}(t, x).$$

5.3 Integration of the dynamic system and events detection

5.3.1 The ode function

The simulator integrates this differential equation with the scilab function `ode` used with the `root` option. This function with this option uses an ordinary differential equation solver with roots' detection. This solver is the **lsodar** solver of the **ODEPACK** package. It is a variable time and order solver: the user specifies the times at which he wants the solution to be computed but in fact the solver computes in internal way a variable optimal step.

The general call to this function with this option is of the form:

```
[x, DetectedEvents, w, iw] = ode('root', x0, t0, T, f, ng, g, w, iw);
```

where:

- the string `root` signifies that ode computes the solution of the differential equation $\dot{x} = f(t, x)$ until the state $x(t)$ crosses the surface $g(t, x) = 0$;
- x_0 is the initial condition;
- t_0 is the initial time;
- T the times at which the solution is computed;
- f the function which define the differential equation;
- g is a function of syntax $y = g(t, x)$ with y a vector of size ng, the goal being to cancel out one of its component;
- w and iw are optionnal vectors for storing information returned by the integration routine. When these vectors are provided in parameters of ode the integration re-starts with the same parameters as in its previous stop;
- $DetectedEvents$ is a vector which gives us the stopping time and which component of g were canceled.

In the `Simulation` function, the differential equation is integrated from each sampling time to the next sampling time and the function g corresponds to the events.
Futhermore, two calls to the `ode` one are made according to the $WarmStart$ flag:

1. If $WarmStart$ is at *False*, either it is the first call to the `ode` function, or changes in actuation (contractile switchs, clock ticks, states or *pwi* changes in `StaticCalciumKneeMuscles` and `StaticCalciumRightKneeMuscles` modules and application of commands in the case of `BipActuators` module) or in contact (lift-off or impacts) appeared since the last handling of events. Then the optionnal parameter w and iw in the `ode` function are not used;

2. If $WarmStart$ is at *True*, neither events were detected at the preceding sampling period nor changes in actuation or contact appeared since the last handling of events. Then we can restart the integration with the same parameters as in its previous stop and the parameter w and iw are used.

If an event is detected during the sampling period, the corresponding component of g reaches or crosses zero. In HuMAnS, the function g is the `EventDetection` function located in the `Kernel` directory.

5.3.2 The EventDetection function

The `EventDetection` function is used by the `ode` function to detect when an event occurs in order to handle it before restarting the integration of the ordinary differential equation. The call to this function is of the form:

```
[Events] = EventDetection(t, x)
```

where x is the system vector state computed by the ode function at time t and $Events$ is a numeric vector containing informations about contact and actuation events (see section Contacts and Actuation State).

This function calls the `ContactEventDetection` and `ActuationEventDetection` functions to detect these events. Then, the $Events$ variable is of the form:

$$Events = \begin{pmatrix} \text{informations about contacts} \\ \text{informations about actuations} \end{pmatrix}$$

Then, if a component of the $Events$ variable reachs or crosses zero, the integration of the ordinary differential equation stops and events are handled.

5.3.3 The Event Handling

If events are detected, the simulator must handle them and call the `ContactEventHandling` or the `ActuationEventHandling` functions according to the type of the events. If the `ContactEventHandling` function is called and if impacts or lift-off appeared, or joints limits were reached (then $WarmStart$ has been set to $False$), the `ActuationStateReset` function is called in order to update the $ActuationState$ variable. For more informations, see the paragraph **??**.

Chapter 6

Contacts and Actuations

6.1 The different states in HuMAnS

The states in HuMAnS are of three types:

- contact state;
- actuation state;
- forces state.

The last type (z variable in HuMAnS) is used only in the case of simulation with muscles model.

The contact state (*ContactState* variable in HuMAnS) describes the state of the different contacts. The contacts are points of contact which can touch their environment but cannot penetrate it or an joint which cannot exceed joints limits.

The actuation state (*ActuationState* variable in HuMAnS) gives informations about everything connected to the actuation as voltage commands in the case of the robot Bip, or impulses to send in muscles in the case of human, or at which times to send these impulses or times to call again the control law...

The contact and the actuation are respectively handled by the `LagrangianDynamics` and the `ActuationModel` modules.

6.1.1 z State in the simulation with muscle model

Currently, the z variable is used only by the *FESSitToStand* and *FESSwingUp* applications.

The z State contains the stiffnesses k_c of the muscles contractile component (see the paragraph **??**) and the forces F_c exerced by these contractile components. If we keep the notations of the paragraph **??**, we have:

$$z = \begin{bmatrix} k_c \\ F_c \end{bmatrix} \begin{matrix} \updownarrow Nmuscles \\ \updownarrow Nmuscles \end{matrix}$$

6.1.2 Contact states in the different applications

Only two types of contacts are implemented in HuMAnS:

- contact between points defined in the lagrangian model and the environment;
- contact in the sense of joint limit.

Contact between points and environment

This type of contact is implemented in the *BipOneStep*, *ChairSitToStand*, *SpaceWalk* and *FESSitToStand* applications. For each contact points, we check if the distance between this contact point and the environment is less than $\sqrt{eps}$. If it is the case, the corresponding component in the $ContactState$ vector will be $True$. If not, it will be $False$. Then, ContactState is a column vector of boolean of size the number of contact. The figure 6.1 shows two positions, one implying contact and an other one with no contact.

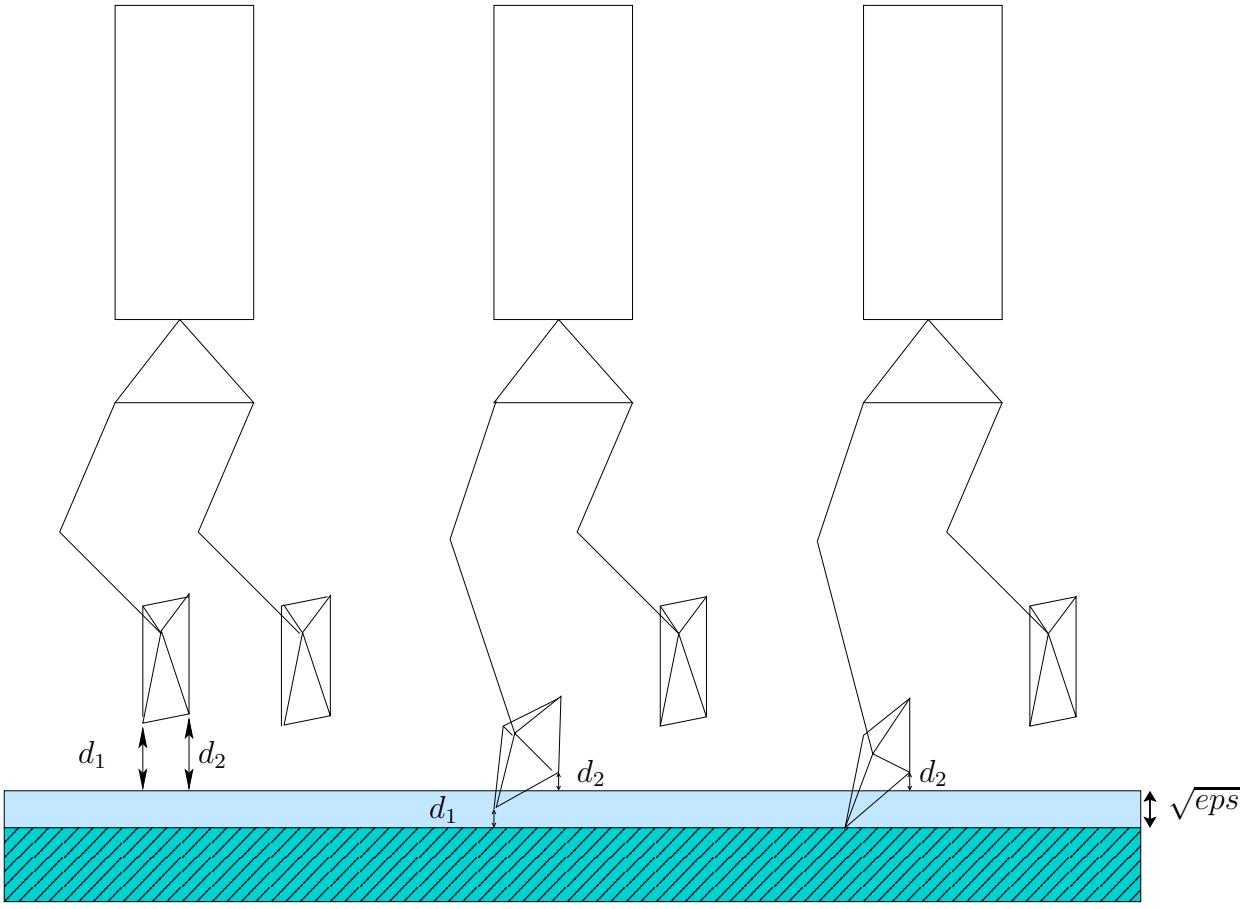


Figure 6.1: There is no contact for the left robot and there is contact for the center and right robot

Contact in the sense of joint limit

These type of contact is implemented in the *FESSwingUp* application. For each articulation,

we see if the body segment is in the limits *ie* the angle between it and its limits positions is greater than $\sqrt{eps}$. If it is the case, the corresponding components in the $ContactState$ vector will be set to $False$. If not, we check for which limit this condition is not fulfilled and the corresponding component in the $ContactState$ vector will be set to $True$ while the one corresponding to the other limit will be set to $False$. The figure 6.2 shows different contacts on the right knee appearing in the *FESSwingUp* application.

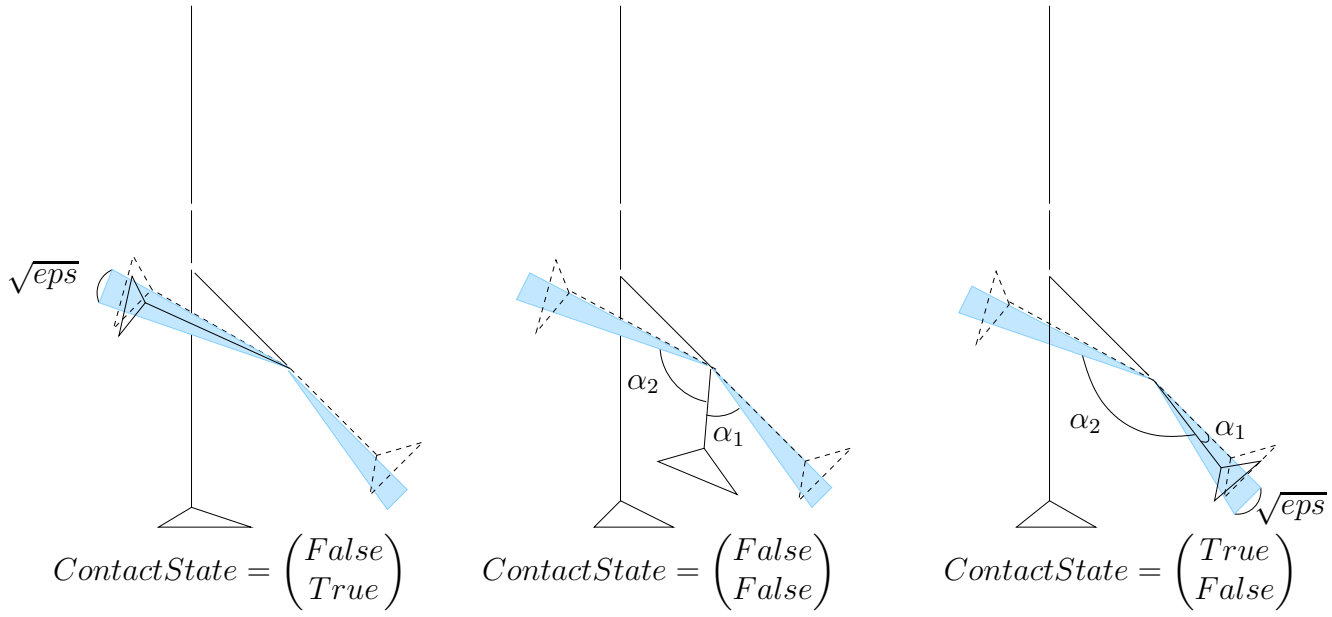


Figure 6.2: There is contact for the left and right human and there is no contact for the center robot

6.1.3 Actuation states in the different applications

ActuationState in NoDynamics module

The `NoDynamics` module is used by the *ChairSitToStand* and the *SpaceWalk* applications. The *ActuationState* variable is a vacuum vector.

ActuationState in StaticCalciumKneeMuscles module

The *ActuationState* variable has a variable size, but is always of the following form:

$$\begin{array}{cc} \left[\begin{array}{c} S_{\dot{\epsilon}_c} \\ State \\ pwi_1 \\ Clock \\ t_1 \\ t_2 \\ pwi_2 \\ t_3 \\ pwi_3 \\ \vdots \\ StoppingTest \end{array}\right] & \updownarrow \begin{array}{l} Nmuscles \\ Nmuscles \\ Nmuscles \\ Nmuscles \\ 1 \\ 1 \\ Nmuscles \\ 1 \\ Nmuscles \\ \\ 1 \end{array} \end{array}$$

The different variables have the following meaning:

- $S_{\dot{\epsilon}_c}$ corresponds to the contractile switchs. It is the sign of ϵ_c and then it tells us if the corresponding muscle is in contraction or in extension. Because it is a sign, it can only take the values 1 (extension) and -1 (contraction). The size of the $S_{\dot{\epsilon}_c}$ vector is the number of muscles. This sign is computed with the following formula:

$$S_{\dot{\epsilon}_c} = \text{sign}(k_s L_0 \dot{\epsilon} + F_c|u| - \alpha F_0 |u|_+)$$

- $State : Exp$ corresponds to the state of the command signal (see figure 1.1). It corresponds to the calcium concentration. Then it can take for each muscle the values 0 (there is no calcium), 1 (the calcium rate decreases) and 2 (the calcium rate is at its maximum). The size of the $State : Exp$ vector is the number of muscles.

- pwi_1 is the product of the width pw and intensity i of the impulse applied at the actual time.

- $Clock$ contains the times at which the calcium rates will change of state. It is a vector of size the number of muscles.

- t_1 is the time at which the control law must be called again.

- The following t_2 and pwi_2 parts are the time t_2 at which to send the impulse described by the pwi_2 product. The time t_2 is of the form $t + \tau$ with t the time at which this impulse has been computed and τ the time needed by this impulse to be really effective. These parts are optionnal and are successively stored and unstored. The t_3 and pwi_3 parts have the same meaning. These optionnal parts form a stack.

- The *StoppingTest* variable allows the simulator to know when the stack ended. Its values is $InitialTime - 1$.

ActuationState in BipActuators module

The *ActuationState* variable in `BipActuators` module is a column vector of size $2 + 2 * NMotors + NDDL$ with $NMotors$ the number of motors, that is the number of joints. The *ActuationState* variable is of the form:

$$ActuationState = \begin{bmatrix} t_1 \\ t_2 \\ Voltage_1 \\ Voltage_2 \\ q \end{bmatrix} \begin{matrix} \updownarrow 1 \\ \updownarrow 1 \\ \updownarrow NMotors \\ \updownarrow NMotors \\ \updownarrow NDDL \end{matrix}$$

The roles of the *ActuationState* parts are the following:

- The first *ActuationState* component t_1 is the time at which the control law must be called again. This call is made every $ControlPeriod$ periods. $ControlPeriod$ is a data given by the user in the application script.
- The second component t_2 is the time at which the previously computed voltage must be applied. In fact, in real experiments, there is a delay between the computation of the command and its effective application.
- The $Voltage_1$ part is the voltage to send to motors at actual time.
- The $Voltage_2$ part stocks the voltage computed to send it to the motors when t_2 is reached, that is to say that a $ControlDelay$ period will be elapsed.
- The q part allows us to compute the gear ratios at each time. But as the gear ratios slow down the simulation a lot, they are computed always with the same position q which corresponds to the initial position.

6.2 Initialization of state variables

The *ContactState* and *ActuationState* variables initialization is made respectively by the `ContactInitialisation` and `ActuationInitialisation` functions.

6.2.1 Contact states initialization in the different applications

The *ContactState* variable is initialized as it is explained in the paragraph 6.1.2.

6.2.2 Actuation states initialization in the different applications

The `ActuationInitialisation` functions initialize not only the *ActuationState* variable but also the z variable.

The call to the `ActuationInitialisation` function is made in the following form:

```
[State, z] = ActuationInitialisation(t, q, qdot)
```

with *State* the *ActuationState* variable.

The general scheme of this initialization is to call the `ActuationEventHandling` function with an *Events* vector set to *False* excepted for the components corresponding to the flag given by the call to the `ControlLaw` function (see below the paragraph 6.4.2 on the `ActuationEventHandling` function). Then an impulse to send to the muscles or a voltage to send to the motors is computed and stored in order to apply it when the delay would be elapsed.

The only differences between the `ActuationInitialisation` functions of all modules are the following:

- In `BipActuators` module, the *ActuationState* is updated a second time after the call to the `ActuationEventHandling` function in order to send the impulsion at initial time (without any delay).
- In `NoDynamics` module, the *ActuationState* variable remains a vaccum vector during the whole simulation. So the `ActuationInitialisation` does nothing.

6.3 Detection of state changes in HuMAnS

In order to deals with the changes of state, we must detect these changes. This detection is made by the `EventDetection` function in `Kernel` directory which calls the `ContactEventDetection` and the `ActuationEventDetection` functions.

6.3.1 Detection of change of contact state in the different applications

The call to the `ContactEventDetection` functions is made in the following form in all modules:

```
[Events] = ContactEventDetection(t, x, ContactState)
```

where *Events* is explained below and t, x and *ContactState* are respectively the actual time, the actual system state and the preceding contact state.

The `ContactEventDetection` functions return a column vector containing

- for each points of contact either the distance between the contact point and its environment increased by $\sqrt{eps}$ if the contact point was not in contact with this environment at the preceding time, or the distance between the contact point and its environment if the contact point was in contact with it at the preceding time. These distances d_1 and d_2 for two contacts points are shown on figure 6.1.
- or, in the case of joints limits, the angle between the articulations and the limits of these articulations if there was contacts at preceding time, or the angle between the articulations and the positions defined by an angle of $\sqrt{eps}$ from the limit positions if there was not contacts at the preceding time. These angles α_1 and α_2 for the knee are shown on figure 6.2.

The size of this column vector is the number of possible contacts.

6.3.2 Detection of change of actuation state in the different applications

The `ActuationEventDetection` functions of all modules (excepted in the `NoDynamics` module which is treated below) return an *Events* vector which has a base part corresponding to the times at which the control law must be called and to the delay before the application of the voltage command.

The call to the `ActuationEventDetection` functions is made in the following form in all modules:

```
[Events] = ActuationEventDetection(t, x, State)
```

where *Events* is explained in the below paragraphs and t, x and $State : Exp$ are respectively the actual time, the actual system state and the preceding actuation state.

Detection of ActuationState changes in NoDynamics module

The *ActuationState* variable remains a vacuum in this module. Then no change appears and the `ActuationEventDetection` function does nothing.

Detection of ActuationState changes in StaticCalciumKneeMuscles and StaticCalciumRightKneeMuscles modules

The *Events* vector is a numeric column vector of size $2Nmuscles + 2$. If t is the actual time and $S_{\dot{\epsilon}_c}$ and $Clock$ the parts of the preceding *ActuationState*, *Events* is given by:

$$Events = \begin{bmatrix} k_s L_{c_0}\dot{\epsilon} + F_c|u| - \alpha F_0|u|_+ + \sqrt{eps}S_{\dot{\epsilon}_c} \\ t - Clock + 10^2 eps \\ t - t_1 \\ (t - t_2) \text{ or } (t - StoppingTest) \end{bmatrix} \begin{matrix} \updownarrow Nmuscles \\ \updownarrow Nmuscles \\ \updownarrow 1 \\ \updownarrow 1 \end{matrix}$$

- The first part of *Events* corresponds to the detection of contractile switches. If the $k_s L_{c_0}\dot{\epsilon} + F_c|u| - \alpha F_0|u|_+$ expression crosses zero, its sign changes and the muscle is contracting or stretching itself;
- The second part corresponds to the detection of clock ticks. If the state of the calcium concentration changes, there is a clock tick;
- The $t - t_1$ part allows the simulator to know if the control law must be called or not;
- The last part corresponds to the unstorage of the stack. If there is not any stack, the *Events* last component value is $t - StoppingTest$ and is always positive and then does not cross zero and no detection is made. If there is a stack, this value is $t - t_1$ and test if the impulse's delay is elapsed or not.

Detection of ActuationState changes in BipActuators module

The *Events* vector is a numeric column vector of size two. If t is the actual time, *Events* is given by:

$$Events = \begin{pmatrix} t - ActuationState(1) \\ t - ActuationState(2) + 10^2 eps \end{pmatrix}$$

The first component corresponds to the $ControlPeriod$ period. It contains the remaining time to go to the time at which the control law must be called again. If this time is reached or exceeded, there is detection of actuation state change.

The second component corresponds to the $ControlDelay$ period. In fact, in real experiments there is a delay between the computation of the voltage and the real application of this voltage command. So the simulator computes at t_1 the command for the time $t_1 + ControlDelay$ and really applies it at time $t_1 + ControlDelay$. The $ActuationState(2)$ is this last time. Then the second component of the Events vector will cross zero when the time will reach this last time and we will have an actuation state change.

6.4 Handling of state changes in HuMAnS

The functions handling the state changes are the `ContactEventHandling` and the `ActuationEventHandling` functions.

6.4.1 Handling of contact state changes in the different applications

The call to the `ContactEventHandling` function in all modules is:

```
[Newx, NewState, WarmStart] = ContactEventHandling(t, x, Events, OldState)
```

where $Newx$ and x are the next and the actual vector state, $NewState$ and $OldState$ are the next and the actual $ActuationState$, t is the actual time and $Events$ is a boolean vector corresponding to the actuation state changes.

The $WarmStart$ variable is changed in the `ContactEventHandling` functions:

- in the `Complete` and `SitToStand` directories, $WarmStart$ is set to $False$ if impacts or lift-offs occurred;
- in the `RightKneeOnly` directory, $WarmStart$ is set to $False$ if the right knee joint reached one of its limits.

The `ContactEventHandling` functions are in the same form in all directories. They update the $ContactState$ variable and if an impact occured, they call the `ImpactLaw` function which computes the velocity after the impact. If this function is very simple, it is implemented in the `ContactEventHandling` function itself as it is the case in the `StaticCalciumRightKneeMuscles` and `StaticCalciumKneeMuscles` directories.

The ImpactLaw function

...

6.4.2 Handling of actuation state changes in the different applications

The call to the `ActuationEventHandling` function in all modules is:

```
[Newx, NewState, WarmStart] = ActuationEventHandling(t, x, Events, OldState);
```

where $Newx$ and x are the next and the actual vector state, $NewState$ and $OldState$ are the next and the actual $ActuationState$, t is the actual time and $Events$ is a boolean vector corresponding to the actuation state changes.

Handling of ActuationState changes in NoDynamics module

The $ActuationState$ variable is always a vacuum vector. So the `ActuationEventHandling` function does nothing.

Handling of ActuationState changes in StaticCalciumKneeMuscles and StaticCalciumRightKneeMuscles modules

As for the actuation state changes detection, the four following cases are treated:

1. *First case:* contractile switches appeared for some muscles.

2. *Second case:* clock ticks appeared for some muscles.

3. *Third case:* a time $NextTime$ is elapsed since the last call of the control law. Then, we must compute the new impulse to send in muscles.

4. *Fourth case:* the delay τ between the computation of the command and its effects on the muscles is elapsed. The impulse must be applied.

First case:

In this case, the simulator must update the sign of $\dot{\epsilon}_c$ in *ActuationState*. Then the components of $S_{\dot{\epsilon}_c}$ corresponding to the muscles that had contractile switches are multiplied by -1. The $Warmstart$ variable is set to $False$

Second case:

In this case, clock ticks appeared. Then the state of the signal command (or of the rate of calcium) changed and the pwi_1 to send to muscles too. Then we must update the $State : Exp$, the pwi_1 and the $Clock$ parts of *ActuationState*. The *ActuationState* becomes:

$$\begin{bmatrix} S_{\dot{\epsilon}_c} \\ \mathbf{State - 1} \text{ for the concerned muscles} \\ \mathbf{pwi_1} * (\mathbf{State - 1}) \text{ for the concerned muscles} \\ \mathbf{t} + \tau_\mathbf{2}(\mathbf{2(State - 1) - 1}) \text{ for the concerned muscles} \\ t_1 \\ t_2 \\ pwi_2 \\ t_3 \\ pwi_3 \\ \vdots \\ StoppingTest \end{bmatrix}$$

with $State : Exp$, $S_{\dot{\epsilon}_c}$, pwi_i $(i = 1, 2, 3)$, t_i $(i = 1, 2, 3)$ and $StoppingTest$ the parts of the preceding *ActuationState*. The changes are in bold.

$WarmStart$ is set to $False$.

Third case:

In this case, a $NextTime$ time is elapsed since the last call to the control law. Then the simulator calls this function to compute a new impulse. The $ControlLaw$ function returns the pwi and $NextTime$ variables. The pwi describes the impulsion and the $NextTime$ is the time to wait before calling again the control law. This new impulse will be applied after a delay τ.

Then it is stored at the end of the stack. The part corresponding to the control of the periodic calls to the control law function is updated too. So the *ActuationState* becomes:

$$\begin{bmatrix} S_{\dot{\epsilon}_c} \\ State \\ pwi_1 \\ Clock \\ \mathbf{t + NextTime} \\ t_2 \\ pwi_2 \\ t_3 \\ pwi_3 \\ \vdots \\ \mathbf{t + \tau} \\ \mathbf{pwi} \\ StoppingTest \end{bmatrix}$$

with t the actual time. The changes are in bold.

$WarmStart$ is set to $True$.

Fourth case:

In this case, the delay to send the impulse is elapsed. Then it must be applied to the concerned muscles. So for the concerned muscles,

- the $State : Exp$ of the signal command becomes 2;
- the pwi_1 corresponding to these muscles are updated with the one contained in the stack;
- the clocks ticks corresponding to these muscles are set to $t + \tau_1$ (with t the actual time) because the signal commands are in state 2.

And this impulse is removed from the stack. So the new *ActuationState* is:

$$\begin{bmatrix} S_{\dot{\epsilon}_c} \\ \mathbf{State = 2} \text{ for the concerned muscles} \\ \mathbf{pwi_1 = pwi_2} \text{ for the concerned muscles} \\ \mathbf{Clock = t + \tau_1} \text{ for the concerned muscles} \\ t_1 \\ t_3 \\ pwi_3 \\ \vdots \\ StoppingTest \end{bmatrix}$$

with t the actual time. The changes are in bold.
If a state or a *pwi* product changes occurred, then $WarmStart$ is set to $True$. If not, $WarmStart$ is set to $False$.

Handling of ActuationState changes in BipActuators module

The `ActuationEventHandling` function treats the two following cases:

1. *first case*: a $ControlPeriod$ time is elapsed since the last call to the control law. So we must compute the new voltage to send to the motors.
2. *second case*: a $ControlDelay$ time is elapsed since the last computing of the command. This last one must be really applied.

First case:

In this case, the simulator compute a new command to send to the motors. It is made in three steps:

1. first, the `BipSensors` function is called to simulate the Bip sensors. This function returns the articular variables q_1(given by the motors) and the six forces on the feet at time t.
2. Then the `ControlLaw` function is called to compute the voltage to send to the motors at time $t + ControlDelay$.
3. Finally, the *ActuationState* is modified:
   - The first component is increased of $ControlPeriod$ to know the next time at which to call the control law;
   - The second component is the actual time increased of $ControlDelay$ to know the next time at which to apply the computed voltage;
   - The $Voltage_1$ part is unchanged. The last command must remain applied for a $ControlDelay$ period;
   - The $Voltage_2$ part is updated with the computed voltage.
   - and the last part of *ActuationState* is unchanged.

Second case:

In this case, the simulator updates the *ActuationState* variable in order to apply the command computed at time $t - ControlDelay$. The *ActuationState* variable is modified as following:

- The second component is decreased of $ControlDelay$;
- The voltage computed at time $t - ControlDelay$ must be applied. Then, the $Voltage_1$ part of $ActuationState$ is updated with this computed voltage;
- The other parts remain unchanged.

The $WarmStart$ variable is modified in the two cases. In the first one, it is set to $True$ and in the second one, to $False$.

6.5 The ActuationStateReset function

In the main procedure of HuMAnS (the `simulation` procedure), when an event on the contact occurs, the `ContactEventHandling` function is called and then the `ActuationStateReset` is called or not depending on the $WarmStart$ flag returned by the `ContactEventHandling` function (see paragraph **??**).

If this flag is at $False$, *ie* if impacts, lift-offs or joints limits reaching occurred, the `ActuationStateReset` is called. If not, this function is not called.
The `ActuationStateReset` function call is of the form:

```
[NewState] = ActuationStateReset(t, x, OldState)
```

where x is the system state vector at the time t, $OldState$ and $NewState$ respectively the old and new actuations state.

The `ActuationStateReset` function does nothing in the `BipActuators` and `NoDynamics` modules.

ActuationStateReset in StaticCalciumRightKneeMuscles and StaticCalciumKneeMuscles modules

A knee joints limits has been reached. Then, the velocity of the knee articulation has changed and then the elongation differential too. Then...

Chapter 7

Position Observer

7.1 Position Observer

The BIP robot does not have any sensors giving him his global position and orientation q_2. An observer was created in order to reconstruct these position and orientation from the articular position q_1 and the feet sensors.

This observer uses the `RightFootObserver`, `LeftFootObserver`, `SuspensionObserver`, `RightFootJacobian`, `LeftFootJacobian` and `SuspensionJacobian` functions. These functions are linked in scilab from C-files of same names. They are generated from the maple files in the `ObserverGeometry/MapleCodeGenerator` repertory.

1. The `RightFootObserver` function returns a vector of size 6 that specify the position of the right foot. This vector is:

$$RightFoot(q) = \begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ y_4 \\ z_1 \end{pmatrix}$$

   where (x_k, y_k, z_k) is the position of the tag k in absolute frame. This vector specifies the complete position of the right foot.

2. The `RightFootJacobian` function gives the jacobian of the `RightFootObserver` function. Then, it returns a matrix of size $6 \times NDDL$.

3. As the `RightFootObserver` function, the `LeftFootObserver` and the `SuspensionObserver` functions specify respectively the complete position of the left foot and the suspension attach. Their values are:

$$LeftFootObserver(q) = \begin{pmatrix} x_{11} \\ x_{12} \\ y_{11} \\ y_{12} \\ y_{14} \\ z_{11} \end{pmatrix} \quad and \quad SuspensionObserver(q) = \begin{pmatrix} x_{22} \\ x_{23} \\ y_{22} \\ y_{23} \\ y_{25} \\ z_{22} \end{pmatrix}$$

   The `LeftFootJacobian` and `SuspensionJacobian` are respectively the jacobians of the `LeftFootObserver` and `SuspensionObserver` functions.

The q_2 computation depends on the state of the contact. It uses the following global variables: $observateur\_q$, $observateur\_droit$, $observateur\_gauche$ and $observateur\_actifs$.

1. *observateur_q* corresponds to the observation of q at the last call of the `Observer` function. The time between two calls is the minimum of `ControlPeriod` and sampling time. So the *observateur_q* is a good initialization to find the new q_2.

2. *observateur_actifs* is a vector of two components. The first one (*resp.* the second one) is the sum of the force on the right (*resp.* left) foot measured by the feet sensors at the last call of the `Observer` function.

3. *observateur_droit* and *observateur_gauche* specify respectively the right and left foot position at the last call of the `Observer` function.

According to the state of the contacts, either one or the other of these global variables are used as reference.

At the first call of the `Observer` function, the global variables do not exist. So, they must be initialized. We suppose that the initial position of the robot is known and is in the air, and then we know the values of *observateur_suspendu*. So we use the suspension as reference to initialize the *observateur_q* variable. The *observateur_actifs* is initialized with the feet sensors and the *observateur_droit* and *observateur_gauche* variables are not initialized.

After this initialization and when the `Observer` function is called again, the global position and orientation q_2 is computed as following. If the right foot is well on the ground, it is used as reference to compute the global position. Indeed, if the right foot is well on the ground, it did not move between the preceding call and the present call and then we can minimize the distance between its preceding position (given by *observateur_droit*) and its present position (given by `RightFootObserver`) to compute the global q. A foot is considered to be well on the ground when the sums of the forces on it at the time the `Observer` function is called and just before this time are both greater than 400??(40N?). If the right foot does not fulfill these conditions, the left foot is used as reference if it fulfills them. And if neither the right foot nor the left one are well on the ground, the robot is considered well in the air and so the suspension is used as reference.

Then, if a foot has just layed on the ground, his reference position (*observateur_(droit or gauche)*) is reinitialized. If it is the first foot to come on the ground, the *observateur_(droit or gauche)* is precomputed: it corresponds to the position $q = 0$. If not, we used the `RightFootObserver` or the `LeftFootObserver` functions to compute the position of the foot.

The *observateur_actifs* is reinitialized only if a foot comes well on the ground or if it rises from the ground.

Finally, the velocity $\dot{q}$ is computed with a simple difference between the global q founded and the one computed at the preceding call to the `Observer` function.

Chapter 8

The Task Function and its inverse

8.1 The Task Function and its inverse

The files concerning the task function are in `ActuationModel/BipActuators/TaskFunctionControl/TaskFunctionDefinition` directory. These files are: **TaskFunction.c**, **TaskJacobian.c** and **TaskNLEffects.c**.

8.1.1 Generation of the C files

These C files are generated by the maple files contained in the `MapleCodeGeneration` directory. The **AdditionnalData.maple**, **DynamicData.maple** and **KinematicData.maple** files are the same than those located in the `LagrangianModel/MapleCodeGeneration` directory.

The **TaskGeneration.maple** file contains three procedures (among others) which generate the C-files **TaskFunction.c**, **TaskJacobian.c** and **TaskNLEffects.c**. These procedures are `TaskFunctionVector`, `TaskJacobianMatrix` and `TaskNLEffects`. They are explained below.

TaskFunctionVector procedure

`TaskFunctionVector()` returns a vector $Q(q)$ of size $NDDL$ and uses the following procedures:

- the `FrameMatrixInPelvisFrame` procedure. `FrameMatrixInPelvisFrame(k)` is the linear mapping from the pelvis frame (frame 6) to the frame k.
- the `TagPositionInPelvisFrame` procedure. `TagPositionInPelvisFrame(k)` is the position of tag k in the pelvis frame.
- the `COMPositionInPelvisFrame` procedure. `COMPositionInPelvisFrame(k)` is the position of the center of mass in the pelvis frame.

The table 8.1.1 gives the result of the TaskFunctionVector procedure. In this table, the notations x_G^P, y_{16}^{RF} and z_{18}^{LF} mean respectively the x coordinate of the center of mass in the pelvis frame (P), the y coordinate of the tag 16 in the frame of the right foot (RF) and the z coordinate of the tag 18 in the frame of the left foot (LF).

In scilab, the TaskFunction is called in the following form:

```
[s] = TaskFunction(q)
```

where s is the position vector of the robot in the task space and q the articular position. These two vectors are column vectors of size $NDDL$.

$Q_{1...3}$	$\{z_G^P - z_5^P, y_5^P - y_G^P, x_G^P - x_5^P\}$	????
$Q_{4...6}$	$\{z_G^P - z_{10}^P, y_{10}^P - y_G^P, x_G^P - x_{10}^P\}$	????
$Q_{7...9}$	$\{x_{16}^{RF} - x_{17}^{RF}, z_{17}^{RF} - z_{16}^{RF}, z_{18}^{RF} - x_{16}^{RF}\}$	Trunk orientation in the right foot frame.
$Q_{10...12}$	$\{x_{16}^{LF} - x_{17}^{LF}, z_{17}^{LF} - z_{16}^{LF}, z_{18}^{LF} - x_{16}^{LF}\}$	Trunk orientation in the left foot frame.
$Q_{13...21}$	$q_{13...21}$	

Table 8.1: Position vector of the robot in the task space.

TaskJacobianMatrix and TaskNLEffects procedures

The TaskFunctionVector procedure gives a vector $Q(q)$. The successive differentials of this vector are:

$$\dot{Q}(q, \dot{q}) = H(q)\dot{q} \tag{8.1}$$
$$\ddot{Q}(q, \dot{q}, \ddot{q}) = H(q)\ddot{q} + h(q, \dot{q}) \tag{8.2}$$

with $H(q) = \partial Q/\partial q$ the jacobian matrix of Q, and $h(q, \dot{q})$ the other terms appearing in the differential.

The `TaskFunctionJacobianMatrix` and the `TaskNLEffects` procedures return respectively the matrix $H(q)$ and the vector $h(q, \dot{q})$.

These procedures are used in the `GenerateTask` function to generate respectively the **TaskJacobian.c** and the **TaskNLEffects.c** files.

In scilab, the `TaskJacobian` and the `TaskNLEffects` functions are called in the following forms:

```
[H] = TaskJacobian(q)
[h] = TaskNLEffects(q, qdot)
```

where H and h are respectively a square matrix of size $NDDL \times NDDL$ and a column vector of size $NDDL$.

8.1.2 The InverseTaskFunction function

The **InverseTaskFunction.sci** function is located in the `\ActuationModel/BipActuators/TaskFunction` directory and computes the inversion of the function `TaskFunction`.

Its call in scilab is of the form:

```
[q, erreur] = InverseTaskFunction(TaskValue)
```

where $TaskValue$ parameter is the position of the system in the task space and q is the position of the system such as $Taskfunction(q) = TaskValue$.

8.2 Trajectory Generation

The **OneStepTrajectoryGeneration.sci** script creates the file **OneStep.traj3** which contains the informations about the trajectory that the robot must follow.

The **OneStepTrajectoryGeneration.sci** script uses the `WriteTrajectoryFile` function to create and to write the file **OneStep.traj3**. We'll see later how is the format `.traj3`.

8.2.1 WriteTrajectoryFile function

The call to the `WriteTrajectoryFile` function is :

```
[] = WriteTrajectoryFile(name, positions, data)
```

where *name* is a string which will be the name of the trajectory file (with its path from HuMAnS repertory), *positions* is a matrix of positions of size ($NDDL \times$ (number of positions)) and *data* is a row vector of size $NDDL$ containing informations about the positions corresponding to the columns of the matrix *positions*.

For example, the *name* is `'ActuationModel/BipActuators/TaskFunctionControl/OneStep'`

The *positions* matrix alternate stable and intermediate positions. The stable ones must be reached with zero velocity and acceleration while the intermediate ones are reached with any velocity and acceleration. The *positions* matrix must begin and finish with stable positions. Then, it is in the form:

$$\begin{pmatrix} \begin{pmatrix} stable \\ position \end{pmatrix} & \begin{pmatrix} intermediate \\ position \end{pmatrix} \begin{pmatrix} stable \\ position \end{pmatrix} & \cdots & \begin{pmatrix} intermediate \\ position \end{pmatrix} & \begin{pmatrix} stable \\ position \end{pmatrix} \end{pmatrix}$$

The *data* vector contains informations about the positions of the *positions* matrix. The component associated with stable positions are the times at which these positions are reached. The ones associated with intermediate positions are informations about the contact state. It can take an integer values between 1 and 5. The table 8.2.1 gives the meaning of these values.

1	simple support right foot
2	simple support left foot
3	double support
4	simple support suspension
5	transition between double support and simple support

Table 8.2: Meaning of the informations about contact.

8.2.2 .traj3 format

A part of the **OneStep.traj3** file is shown on figure 8.2.2. The "`Nombre de points`", "`Temps et contacts`" and the "`Coordonnees points`" informations are respectively the number of positions in the *positions* matrix above, the *data* vector above and the transposition of the *positions* matrix.

8.2.3 ReadTrajectoryFile function

The `ReadTrajectoryFile` function returns the *positions* matrix and the *data* vector above. It is called by:

```
[positions, data] = ReadTrajectoryFile(nom)
```

The `ReadTrajectoryFile` and the `WriteTrajectoryFile` are respectively the results of the linking of the `lecture_traj3` and `ecriture_traj3` function in the **TrajectoryFilesTools.c** file.

```
Trajectoire format traj3
Nombre points
33
Temps et contacts
0 4 1 5 2 3 3 3 4.2 5 4.6 1 5.8 5 6.2 3 8.6 5 9 2 10.2 5 10.6 3 11.8 3 12 3 13 5 14 4 15
Coordonnees points
0 0.787 -0.11 0 0.787 0.11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0.802 -0.11 0 0.802 0.11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0.817 -0.11 0 0.817 0.11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0.817 -0.11 0 0.817 0.11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
.
.
.
0 0.817 -0.11 0 0.817 0.11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0.802 -0.11 0 0.802 0.11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0.787 -0.11 0 0.787 0.11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 8.1: Part of the OneStep.traj3 file

Chapter 9

Reconstruction from Optical Sensors

9.1 Introduction

After a quick *Getting Started* to show an example of the reconstruction from optical sensors (section 9.2), an overview of the theory of the reconstruction from optical sensors is made in section 9.3. Then, a global view of the functions used in the reconstruction process is made in the section 9.4. The model used in the reconstruction process is described in section 9.5. Finally, all the scripts used by the reconstruction process are explained in the section 9.6.

9.2 Getting started

The following instructions allow the user to quickly launch an example of a movement reconstruction from optical sensors. A movement is defined as a succession of postures.

1. Open Scilab, go to the `HuMAnS/Tools/Reconstruction/OpticalSensors` directory and launch the **MenuReconstruction** script:

   ```
   > chdir('PathOpticalSensorsDirectory')
   > exec MenuReconstruction.sci;
   ```

   where `PathOpticalSensorsDirectory` is the absolute path of the `HuMAnS/Tools/Reconstruction/OpticalSensors` directory.

   A menu window appears. The figure 9.1 shows this menu window.

2. Then, click on the "**Launch the reconstruction**" button. The reconstruction of a movement contained in a data file is launched. The figure 9.2 shows the scilab window.

   After a while, the reconstructed trajectory (that is to say the succession of postures composing the movement) is compared to the measured 3D positions of the set of markers used by an animation on a graphic window (figure 9.3). These 3D positions have been linked by straight lines in order to make the visualization easier.

9.3 Reconstruction from optical sensors theory

After an overview of the reconstruction process, we will remind some notations used. Then, we will explain the theory of the reconstruction process.

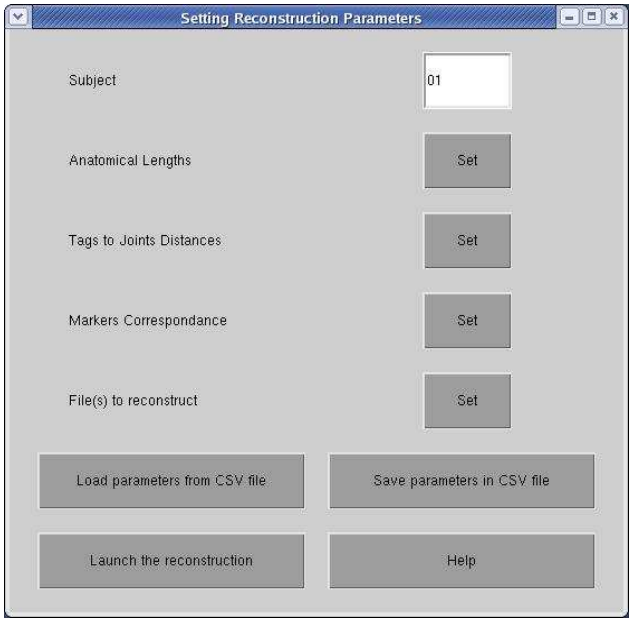


Figure 9.1: Menu window for the reconstruction

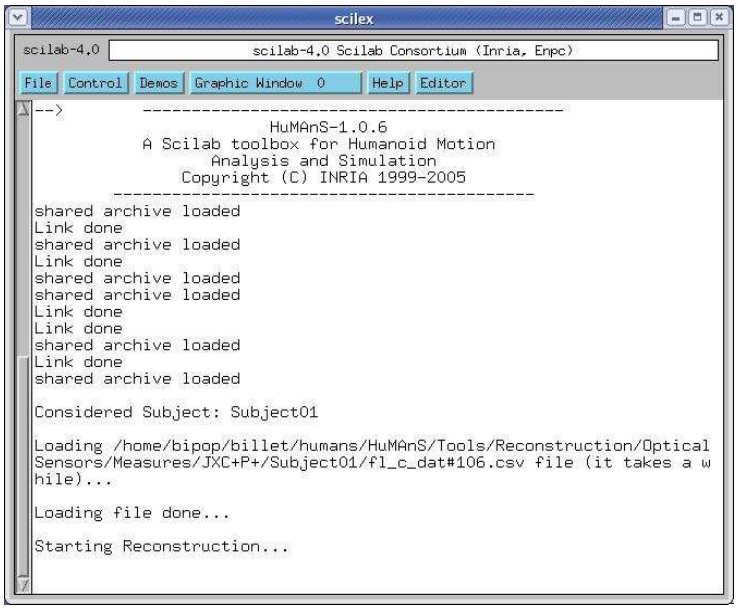


Figure 9.2: Execution of the MenuReconstruction.sci script

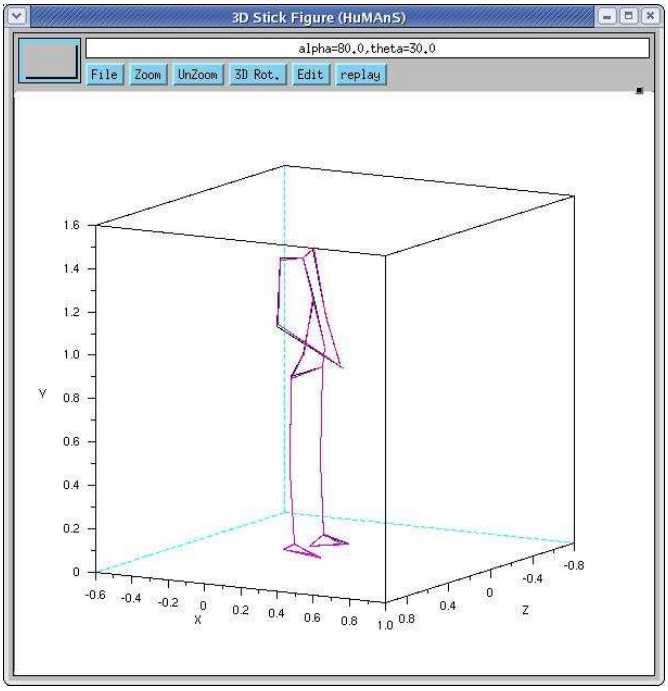


Figure 9.3: Comparison between the reconstructed trajectory (purple model) and the observed 3D positions of the set of markers linked by straight lines (black model). We see only one posture on this window but what we would really get on screnn is an animation drawing the successive postures composing the movement, that is why we talk about trajectory.

9.3.1 About the reconstruction

Only one category of motion sensors have been modeled: optical sensors such as Vicon^{TM} or Optotrack^{TM} devices[1]. But the toolbox provides algorithms to derive new models of motion sensors if necessary.

The optical sensors are supposed to give the 3D position of a set of markers attached to different parts of the body. Therefore their ouptut depend solely on the position and orientation of the different joints of the body. The **Human36** model implemented in the `Reconstruction` module proposes 28 canonical markers, as shown on the figures **??** and **??**, which are sufficient to reconstruct completely the posture. These canonical markers are placed on some anatomical landmarks (fifth metatarsal, great trochanter for example) corresponding to the **Tags** of the **Human36** model.

[1]and others devices such as Coda System, Motion Analysis System, Cellspots...

In the case of optical sensors, the whole set of markers allows to deduce the complete posture of the body. The reconstruction process find the posture for which the computed 3D position of the markers "matches as best as possible" the measured 3D position of the markers. A simple way to do this comparison is to represent the reconstruction process as a general nonlinear least-squares problem. Powerful algorithms such as the FSQP algorithm [**?**] can efficiently solve such nonlinear problems and propose reliable reconstructions even if some of the measures are missing, for example because of occlusions.

The reconstruction is static at the moment. That is to say that each 3D posture of the body during the movement are reconstructed separately considering that the body is not moving. The succession of these 3D postures is named *trajectory* in the following sections.

The `OpticalSensors` directory contains scripts allowing us to reconstruct statically a trajectory from optical sensors.

The reconstruction from optical sensors uses the same kinematic and dynamical model used for the simulations and defined in the `HuMAnS/LagrangianModel/Human36` directory. I advise you to read the chapter 3 in order to understand the Human36 model.

9.3.2 Notations

Some specific words used in the following sections are explained below.

Reference frame or absolute frame

The reference frame is defined in the section 3.1.1. In the case of the **Human36** model, the origin of this frame is the vertical projection on the ground of the midpoint between the two ankles. The X axis is pointing forward, the Y axis is pointing upward and the Z axis is pointing to the right of the body.

Segment frames

A segment frame is the frame attached to a segment, that is to say that the segment does not move in this frame. These frames are defined in the zero-position. The **Human36** model is in the zero-position when the body is standing up, with the feet flat on the ground, spaced apart about the same distance as the width of the hips and with the arms straight and parallel to the sides of the body with the palms of the hands facing to the front of the model. In this position, the origin of the segment frames are on the joints of the segments they are attached to, and they have the same orientations absolute frame. For example, the origin of the frame

attached to the right calvicle is the right sternoclavicular joint and its orientation is the same as the absolute frame, that is to say with X axis pointing forward, Y axis pointing upwards and Z axis pointing to the right of the body. The origins of the segments frames are given in the table 3.1.

Posture or Position:

This is the name of the vector q defined in the section 3.1.1. This vector contains the articular position and the global translation and rotation of the body relatively to the reference frame. The articular position components correspond (with the sign + or −) to the Cardan angles defining the orientation of a segment frame relatively to its parent frame. The figure 9.4 shows a knee joint defining a component of this vector q.

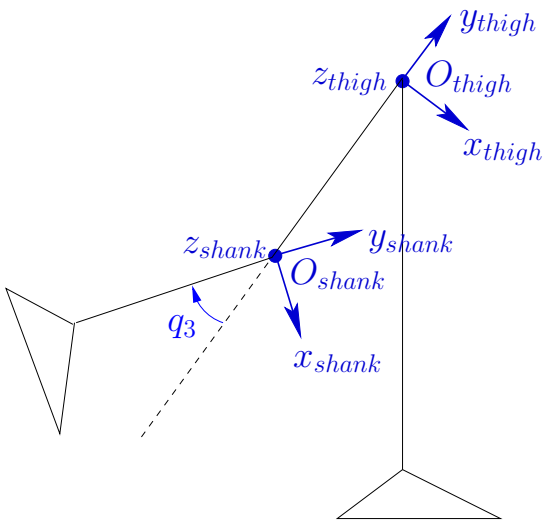


Figure 9.4: Illustration of the rotation of the knee joint. The frame $(O_{thigh}, x_{thigh}, y_{thigh}, z_{thigh})$ and $(O_{shank}, x_{shank}, y_{shank}, z_{shank})$ are respectively the thigh attached frame and the shank attached frame

Trajectory

We call a trajectory a succession of positions q.

Anatomical lengths

The anatomical lengths are the lengths between the differents joints of the body. They are described in the section 3.1.2. As it is explained in the chapter 3 (to read carefully!) the lengths

of the **Human36** model are user-sizibled. The user can get and set in scilab these anatomical lengths with the functions `GetAnatomicalLengths` and `SetAnatomicalLengths` respectively.

Tags

The Tags are defined in the section 3.2. In the case of the **Human36** model, the tags are some anatomical landmarks such as acromion, lateral malleolus... We suppose that the optical markers (the leds) are placed on these tags.

Tag to Joint distances

The tags are placed on some anatomical landmarks. Each tag is attached to a segment of the body, that is to say that the position of this tag in the frame attached to its segment is always the same. The corresponding between the tags and the segment to is defined in table **??**. The default positions (x, y, z) of these tags are defined in the **Human36** model, but the user can set and get these positions in order to match as best as possible to the studied subject. This is made respectively with the `SetTag2JointLengths` and `GetTag2JointLengths` functions.

Model and Subject

When the model is talked about, the **Human36** model is always meant, it is the only one used reconstruction process at this time. The Subject is the real human used in the experiment. To make the reconstruction work well, the user must change the default anatomical lengths and tags to joints distances of the **Human36** model for the lenghts and distances measured on the subject.

9.3.3 Optimization problem

To be completed...

9.4 Global view of the Reconstruction module

The Reconstruction module contains two directories: the **KinematicModel** and the **OpticalSensors** directories. The **KinematicModel** directory allows the user to build the different functions used by the model. The **OpticalSensors** directory contains scripts which use the functions generated in the **KinematicModel** directory to reconstruct positions or trajectories. The data sets are stored in the **OpticalSensors/Measures** directory.

The two following sections describe these two directories.

9.5 KinematicModel directory

The kinematic models used for the reconstruction process are implemented in the **HuMAnS/Tools/Reconstruction/KinematicModel** directory. At the moment, only one kinematic model is implemented: the Human36 one.

If you want to add a new model to the reconstruction, you must create a new directory (which name is the model name) under the KinematicModel one. The new model would be implemented in this new directory.

9.5.1 The KinematicModel/Human36/MapleCodeGeneration directory

This directory contains the **OpticalSensorsModelGeneration.maple** and the **GenerateOpticalSensorsModel.mw** files. The **OpticalSensorsModelGeneration.maple** file implements maple algorithm to create the following C files:

- **TagsJacobian.c**. This file allows to compute the jacobian of the tags vector. After its load in scilab, the user can call the scilab TagsJacobian function in the form:

  ```
  > J = TagsJacobian(q);
  ```

  This function returns the jacobian of the tags vector J. Then this matrix has $NTAGS*3$ rows and $NDDL$ columns.

- **COM.c**. The **COM** function allows the user to get the centers of mass of the different segments of the model and of the global model in the model reference frame. After its load in scilab, the call of this function is of the form:

  ```
  > CentersOfMass = COM(q);
  ```

  where q is the articular position and $CentersOfMass$ is a matrix of $(NSegments + 1)$ rows and 3 columns (x, y and z coordinates). The numbering of the different segments of the Human36 model is given in the table 3.6. The row i of the $CentersOfMass$ matrix is the center of mass of the segment i. The last row is the global center of mass.

  The segments masses and the local position of the segments centers of mass (that is to say the position of a segment center of mass in the frame attached to this segment) are given by anthropometric tables. See the documentation of the Human36 model (chapter 3) to get more details.

9.5.2 The VRML visualization

The **CreateVRMLWithMarkers.c**, **CreateVRMLWithMarkers.h** and the **SomeDefinitions.sci** files allow us to create a vrml animation file which allow to visualize the reconstructed movement of the model and the measured positions of the leds. The figure 9.5 shows an instant frame of a vrml animation created with the `CreateVRMLWithMarkers` function.

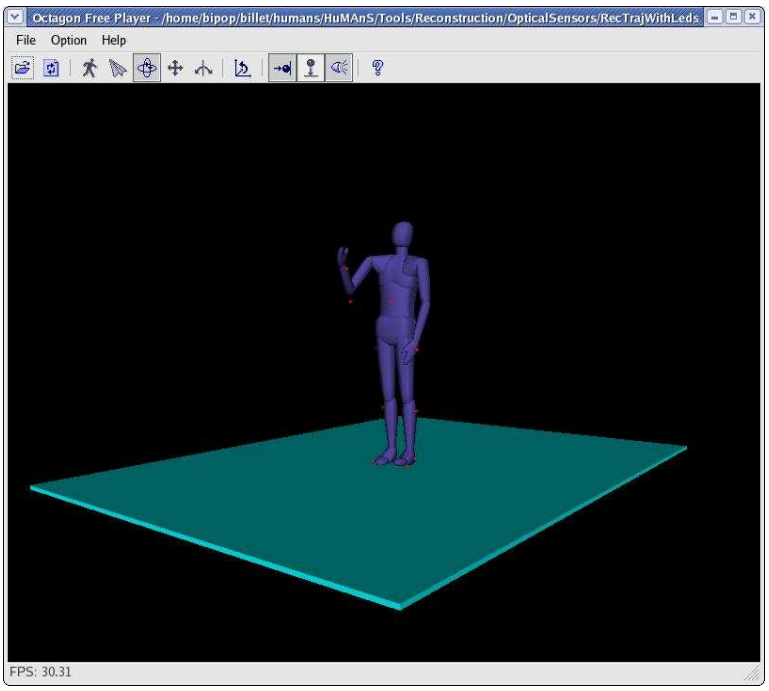


Figure 9.5: Instant frame of a vrml animation created with the **CreateVRMLWithMarkers** function

The call to the `CreateVRMLWithMarkers` function in scilab is of the form:

```
CreateVRMLWithMarkers(deltat, QREC, HAnimJointsNames, ModelName,...
   ModelResolution, OutputFile, TOBS, HAnimMarkersNames, HAnimMarkersRGB);
```

where `deltat` is the sampling time of the trajectory, `QREC` is the reconstructed trajectory, `TOBS` is the matrix of the measured tags, `ModelName` is the name of the model (for example `'Human36'`), `ModelResolution` is the resolution of the model (in the case of the Human36 model, the resolution is `'High'`), `OutputFile` is the name of the output file containing the vrml animation created and `HAnimJointsNames`, `HAnimMarkersNames` and `HAnimMarkersRGB` are constant tables defined in the **SomeDefinitions.sci** file.

9.6 OpticalSensors directory

Firstly, we will describe the set of data acquisitions available in the HuMAnS toolbox. Then, we will see the different scripts used by the reconstruction process.

9.6.1 Set of Aquisition Data

Measurements[2] were made with an Optotrak 3020 in order to test our reconstruction algorithm. The data files correspond to a movement sequence. In these experimentations, the standing subject had the right hand in a cylinder located right ahead. The subject must not touch the border of the cylinder. An unexpected postural disturbance was imposed on the high part of the back with a charge impulsion applied from back to ahead with a balancer (... a little push toward ahead at the level of shoulders). The subject had to maintain its stability (without touching the border of the cylinder). The charge impulsion can be high or low and the cylinder can be small (high precision) or bigger (low precision). Leds are placed on the anatomical landmarks defined in the model description (tags 1 to 18) and on others landmarks specified in the README file contained in the corresponding measures repertories. These other landmarks are not used for the reconstruction. The data files contain the 3D leds positions (x, y, z) in the Optotrak frame. These data files have all the following name: `fl_c_dat#XXX.csv` where `XXX` is the experiment number.

The `Measures` directory under `OpticalSensors` contains the data files described above. The `Measures` repertory has 5 subdirectories which are:

- `JXC(+ or -)P(+ or -)` : These repertories contain the data files corresponding to the different experienments conditions. The `C(+ or -)` correspond respectively to a high or low charge impulsion and the `P(+ or -)` correspond respectively to a high or low precision;
- `ResultsMeasures` : This repertory contains the results of the reconstruction done from the data contained in the `Measures` directory.

In the `JXC(+ or -)P(+ or -)` directories, the subjects are separated. The data files are dispatched in the corresponding `SubjectXX` repertories where `XX` is the subject number.

There is the same directories organization in the `ResultsMeasures` repertory as in the `Measures` one. The files created by the reconstruction process (`RecPosBin`, `RecPosXXX`, `TagPosXXX`

[2]These measurements were made by Olivier Martin, Université Joseph Fourier, Grenoble, France. For more informations: olivier.martin@ujf-grenoble.fr

and `COMPosXXX`) are described in the section 9.6.3.

The figure 9.6 shows this repertory organization.

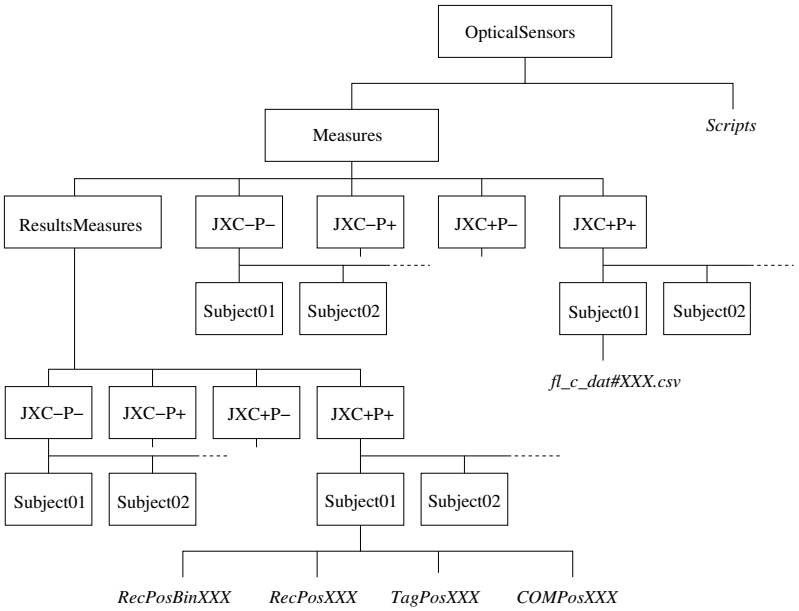


Figure 9.6: OpticalSensors repertory organization

9.6.2 The MenuReconstruction.sci script

This script allows the user to set or load all the parameters required by the reconstruction, to save these parameters in **.csv** files and to launch the reconstruction process. The default parameters are thoses corresponding to the `Subject01` subject. These default parameters are in the **ReconstructionParameters.csv**.

The parameters settings

The figure 9.1 shows the window menu generated by the script. We can see that this window menu asks the user to set five types of parameter:

- **the subject number**. The reconstruction process needs the name of the subject. This name is of the form **SubjectXX** where **XX** is the number of the subject(for example **01**, **02**, ... **10**, ...).
  But it is possible that the user wants to test different parameters (anatomical lengths or

tags to joint distances) for the same subject. The user can do that by setting a number such as **XXYY**...**Y** in the Subject field, where **XX** is the number of the subject and **YY**...**Y** is the number of the test for the subject(for example **1**, ..., **10**, ...). Then, the reconstruction process will consider the subject **SubjectXX** with the parameters corresponding to the test **YY**...**Y** of the subject **XX**. If the parameters are saved in a csv file, the number of the subject put in the Subject Number field is **XXYY**...**Y**.

- **the anatomical lengths of the subject**. The reconstruction process uses a model with some anatomical lengths. These lengths can be modified by the user. Then the user can set these anatomical lengths by clicking on the **Set** button corresponding to the anatomical lenghs in the zero position. A window appears and the user can change the anatomical lengths of the model (to make the model correspond to the subject considered). This window is shown on the figure 9.7. The name of the lengths are specified on the left of the window, in order to facilitate their settings. The figures 3.2 and 3.3 show the lengths to set and the zero-position of the model.

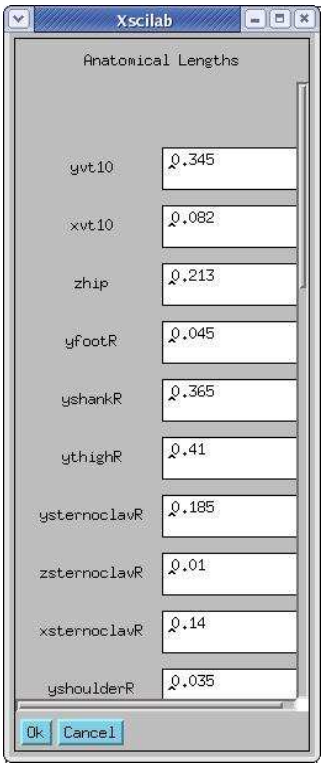


Figure 9.7: Snapshot of the window (on unix) which allows the user to set the anatomical lengths. On windows, it is a little different, but same notation is used.

- **the distances from tags to joints**. The user must set the position (x, y, z) of the leds (placed on some anatomical landmarks called tags) in the frame of the corresponding joint.
  The origins of the joints frames are the joints and the orientations of these frames are the same as the reference frame, that is to say with the x axis pointing to front of the model, the y axis pointing upward and the z axis pointing to the right of the model. The figures 3.5 and 3.6 show the different tags included so far in the Human36 model. The table **??** shows for each tag, the joint that is the origin of the frame in which the position of the tag must be given.
  The figure 9.8 shows the window appearing when the user clicks on the button **Set** corresponding to the tags to joint distances. To each line correspond a different tag. The names are specified on the left of the window. The columns correspond to the position (x, y, z) of the tags in the frame of their corresponding joints.

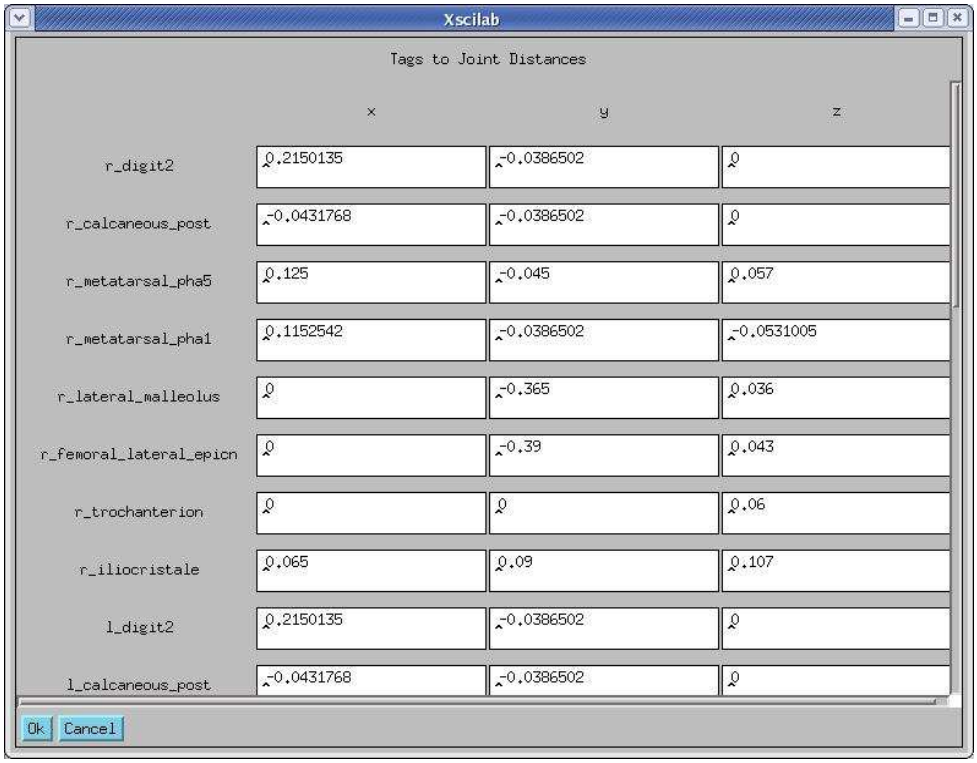


Figure 9.8: Snapshot of the window (on unix) which allows the user to set the positions of the tags in the corresponding joints frames. On Windows systems, it is a little different, but we the same notation is used.

- **the markers correspondance**. The Markers correspondance is a vector of integers. The component i of this vector is:
  - -1 if the user has not put a led on the tag i (that is to say if there is no data corresponding to the tag i in the csv file containing the measures).
  - j if the user has put a led on the tag i. j is the number of this led in the measured data.

  The figure 9.9 shows the window which appears if the user clicks on the **Set** button corresponding to the Markers correspondance. The names on the left of the figure are the names of the tags in the Human36 model.

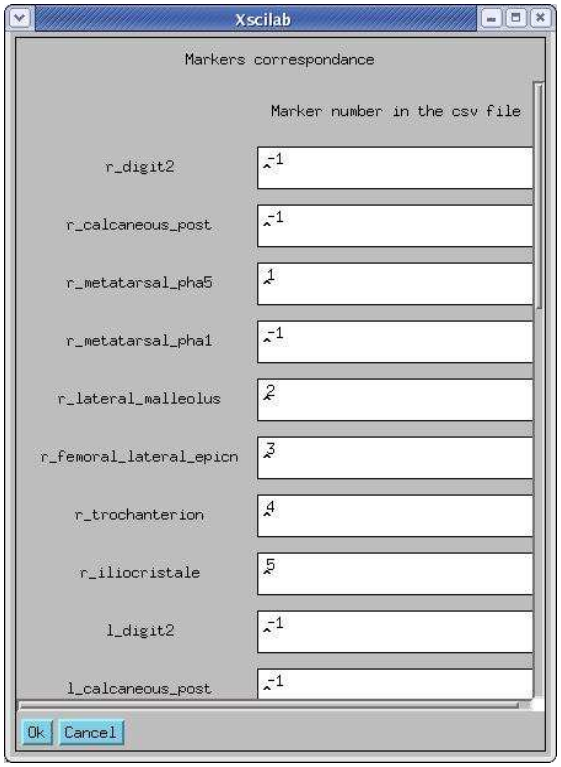


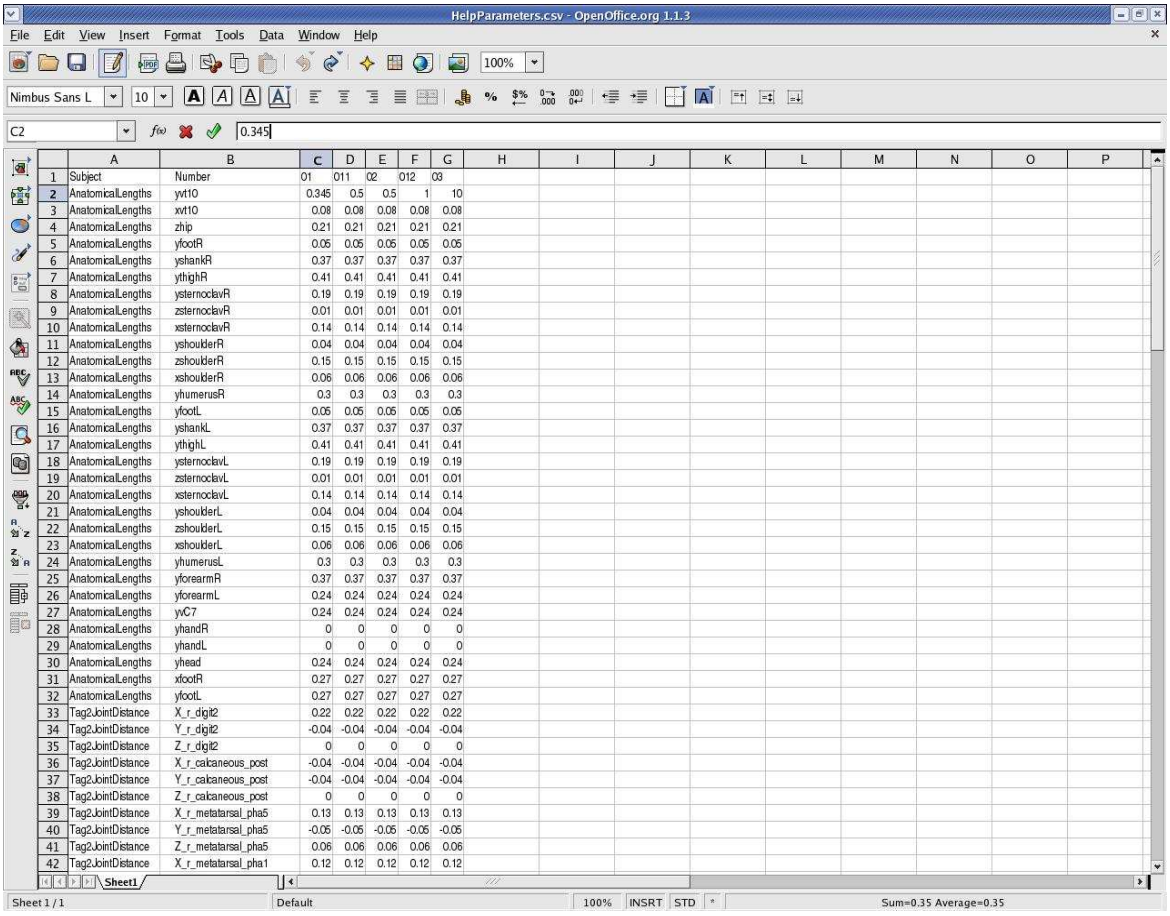
Figure 9.9: Snapshot of the window (on unix) which allows the user to set the markers correspondances. On windows, it is a little different, but the same notation is used.

- **the file(s) to reconstruct**. The user can decide to reconstruct all the data files contained in the `Measures/JXC(+ or -)P(+ or -)/SubjectXX` directories, were `XX` is the number given by the user for the `Subject number` parameter. If the user does not set these parameters, the file to reconstruct is the `Measures/JXC+P+/Subject01/fl_c_dat#106.csv`.

The Load/Save parameters from/in .csv files

The user can load (or save) some parameters from (or in) .csv files. These parameters are the number of the subject (in the form **XXYY**...**Y**, see the precedent paragraph), the anatomical lengths and the tag to joint distances.

An example of a csv file of parameters is represented on the figure 9.10. The parameters are stored in columns, each column corresponds to a subject. If the user want to save a set of parameters in an existing csv file of parameters, the new parameters are saved at the end of this file, in a new column.



	A	B	C	D	E	F	G
1	Subject	Number	01	011	02	012	03
2	AnatomicalLengths	yvt10	0.345	0.5	0.5	1	10
3	AnatomicalLengths	xvt10	0.08	0.08	0.08	0.08	0.08
4	AnatomicalLengths	zhip	0.21	0.21	0.21	0.21	0.21
5	AnatomicalLengths	yfootR	0.05	0.05	0.05	0.05	0.05
6	AnatomicalLengths	yshankR	0.37	0.37	0.37	0.37	0.37
7	AnatomicalLengths	ythighR	0.41	0.41	0.41	0.41	0.41
8	AnatomicalLengths	ysternoclavR	0.19	0.19	0.19	0.19	0.19
9	AnatomicalLengths	zsternoclavR	0.01	0.01	0.01	0.01	0.01
10	AnatomicalLengths	xsternoclavR	0.14	0.14	0.14	0.14	0.14
11	AnatomicalLengths	yshoulderR	0.04	0.04	0.04	0.04	0.04
12	AnatomicalLengths	zshoulderR	0.15	0.15	0.15	0.15	0.15
13	AnatomicalLengths	xshoulderR	0.06	0.06	0.06	0.06	0.06
14	AnatomicalLengths	yhumerusR	0.3	0.3	0.3	0.3	0.3
15	AnatomicalLengths	yfootL	0.05	0.05	0.05	0.05	0.05
16	AnatomicalLengths	yshankL	0.37	0.37	0.37	0.37	0.37
17	AnatomicalLengths	ythighL	0.41	0.41	0.41	0.41	0.41
18	AnatomicalLengths	ysternoclavL	0.19	0.19	0.19	0.19	0.19
19	AnatomicalLengths	zsternoclavL	0.01	0.01	0.01	0.01	0.01
20	AnatomicalLengths	xsternoclavL	0.14	0.14	0.14	0.14	0.14
21	AnatomicalLengths	yshoulderL	0.04	0.04	0.04	0.04	0.04
22	AnatomicalLengths	zshoulderL	0.15	0.15	0.15	0.15	0.15
23	AnatomicalLengths	xshoulderL	0.06	0.06	0.06	0.06	0.06
24	AnatomicalLengths	yhumerusL	0.3	0.3	0.3	0.3	0.3
25	AnatomicalLengths	yforearmR	0.37	0.37	0.37	0.37	0.37
26	AnatomicalLengths	yforearmL	0.24	0.24	0.24	0.24	0.24
27	AnatomicalLengths	yvC7	0.24	0.24	0.24	0.24	0.24
28	AnatomicalLengths	yhandR	0	0	0	0	0
29	AnatomicalLengths	yhandL	0	0	0	0	0
30	AnatomicalLengths	yhead	0.24	0.24	0.24	0.24	0.24
31	AnatomicalLengths	xfootR	0.27	0.27	0.27	0.27	0.27
32	AnatomicalLengths	yfootL	0.27	0.27	0.27	0.27	0.27
33	Tag2JointDistance	X_r_digit2	0.22	0.22	0.22	0.22	0.22
34	Tag2JointDistance	Y_r_digit2	-0.04	-0.04	-0.04	-0.04	-0.04
35	Tag2JointDistance	Z_r_digit2	0	0	0	0	0
36	Tag2JointDistance	X_r_calcaneous_post	-0.04	-0.04	-0.04	-0.04	-0.04
37	Tag2JointDistance	Y_r_calcaneous_post	-0.04	-0.04	-0.04	-0.04	-0.04
38	Tag2JointDistance	Z_r_calcaneous_post	0	0	0	0	0
39	Tag2JointDistance	X_r_metatarsal_pha5	0.13	0.13	0.13	0.13	0.13
40	Tag2JointDistance	Y_r_metatarsal_pha5	-0.05	-0.05	-0.05	-0.05	-0.05
41	Tag2JointDistance	Z_r_metatarsal_pha5	0.06	0.06	0.06	0.06	0.06
42	Tag2JointDistance	X_r_metatarsal_pha1	0.12	0.12	0.12	0.12	0.12

Figure 9.10: Example of a csv file of parameters. This file contains 5 sets of parameters

The "Launch the reconstruction" button

The "Launch the reconstruction" button launchs the **StaticTrajectoriesReconstruction.sci** script with the default or user set parameters.

9.6.3 The StaticTrajectoriesReconstruction.sci script

The **StaticTrajectoriesReconstruction** script makes the following loop as shown on figure 9.11:

1. it checks if they is any data files which is not reconstructed in the directory corresponding to the given subject (that is to say in the `Measures/JXC(+ or -)P(+ or -)/SubjectYY`). If yes it considers one of these data files (`fl_c_dat#XXX.csv`) to reconstruct the corresponding trajectory and then it makes the following step. If all trajectories have already been reconstructed, it stops the script;

2. it converts the format of the trajectory data contained in the **.csv** file choosen at the preceding step (with he **ConvertFromCsvToTags** function explained below);

3. it reconstructs statically this trajectory. For each position of the trajectory, it calls the **Reconstruction** function which is explained below. This function returns the corresponding reconstructed position;

4. Then all the positions of the trajectory were reconstructed and stacked in the $QREC$ matrix. These matrix has $NDOF$ rows and $NSamplings$ columns. $NDOF$ is the degrees of freedom of the model (33 for Human27 model) and $NSamplings$ is the number of samplings of the trajectory (that is to say the number of successive positions in the trajectory). Then $QREC$ is of the form:

$$QREC = \begin{pmatrix} q_{1,1} & \cdots & q_{1,\ NSamplings} \\ \vdots & \vdots & \vdots \\ q_{NDOF,1} & \cdots & q_{NDOF,\ NSamplings} \end{pmatrix}$$

   where $q_{i,j}$ corresponds to the position component i at the sampling j.

5. Then it saves some data in different formats in the corresponding directory, that is to say in the `ResultMeasures/JXC(+ or -)P(+ or -)/SubjectYY` directory. The data saved are:

- the successive reconstructed positions matrix $QREC$. This data is saved in binary and text format. The $QREC$ matrix is saved in binary format in the `RecPosBinXXX` file which is placed in the directory corresponding to the input data file (same subject, same experiences conditions and same number `XXX`). It is saved in `.csv` format too in the `RecPosXXX` file which is placed under the same directory. The `.csv` format is described below;
- the tags (or leds) positions in the model reference frame. These data are saved in `.csv` format in the `TagPosXXX` file which is placed under the same directory as the `RecPosBinXXX` and `RecPosXXX` files.
- the positions of the centers of mass of each model segment in the model reference frame. These data are saved in `.csv` format in the `COMPosXXX` file which is placed under the same directory as the `RecPosBinXXX`, `RecPosXXX` and `TagPosXXX` files.

In the `.csv` format, the rows correspond to the samples and the columns to the saved variables:

- for the `QREC` data, the variables are the articular position components:

$$(q_1, q_2, \ldots, q_{33})$$

- For the tags positions, the variables are the tags and center of mass coordinates:

$$(X_1, Y_1, Z_1, \ldots, X_{NTags}, Y_{NTags}, Z_{NTags}, X_{GlobalCOM}, Y_{GlobalCOM}, Z_{GlobalCOM})$$

  The numerotation of the tags is the same as the one given in section 3.2.

- For the segments centers of mass, the variables are their coordinates:

$$(X_1, Y_1, Z_1, \ldots, X_{NSegments}, Y_{NSegments}, Z_{NSegments}, X_{GlobalCOM}, Y_{GlobalCOM}, Z_{GlobalCOM})$$

  The numerotation of the centers of mass is the same as the one given in the section 9.5.1.

Then the first row corresponds to the names of the variables (`QREC`i for the $QREC$ matrix, `X`i, `Y`i, `Z`i for the tags and centers of mass positions. The first column corresponds to the sampling numbers.

6. Then the script go back to step 1.

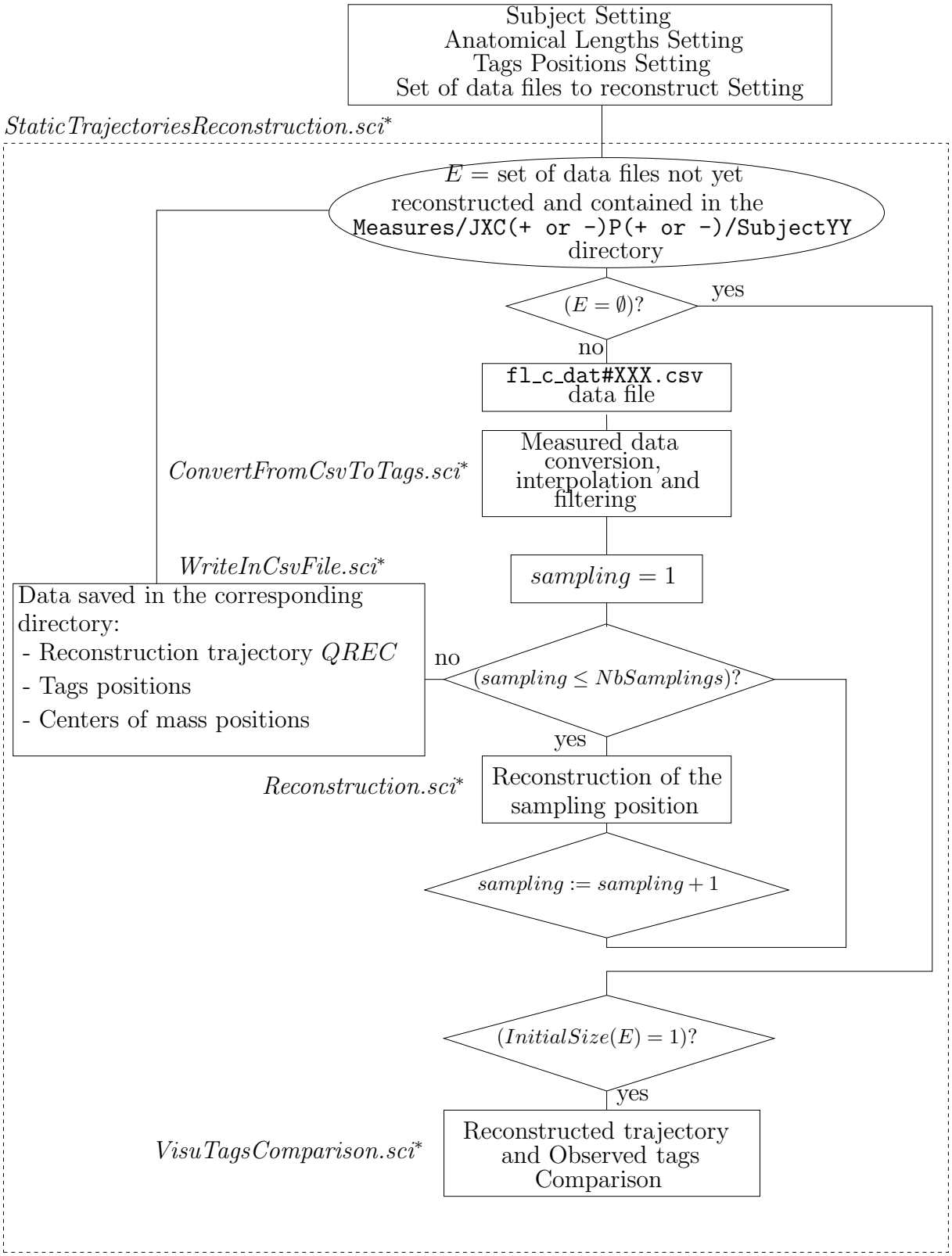


Figure 9.11: Global view of the reconstruction functionnement. The italic names (*) correspond to different scripts and function. The **ConvertFromCsvToTags** function is detailed in the figure 9.12.

9.6.4 The data conversion and filtering: the ConvertFromCsvToTags function

The global view of the **ConvertFromCsvToTags** function is shown on figure 9.12. It allows the user to convert the data containing in the `fl_c_dat#XXX.csv` files in the format required for the reconstruction function and to filter or to interpolate them.

The input parameter is the name of the file containing the trajectory to reconstruct. The returned data are:

- `TOBS`: it is the matrix of the measured positions of the tags used in the reconstruction. These positions are given in the model reference frame. The tags used in the reconstruction are the tags 1 to 18. `TOBS` is of the following form:

$$TOBS = \begin{pmatrix} \begin{pmatrix} x^0_{tag_1,1} & y^0_{tag_1,1} & z^0_{tag_1,1} \\ x^0_{tag_2,1} & y^0_{tag_2,1} & z^0_{tag_2,1} \\ \vdots & \vdots & \vdots \\ x^0_{tag_{18},1} & y^0_{tag_{18},1} & z^0_{tag_{18},1} \end{pmatrix} & \cdots & \begin{pmatrix} x^0_{tag_1,NS} & y^0_{tag_1,NS} & z^0_{tag_1,NS} \\ x^0_{tag_2,NS} & y^0_{tag_2,NS} & z^0_{tag_2,NS} \\ \vdots & \vdots & \vdots \\ x^0_{tag_{18},NS} & y^0_{tag_{18},NS} & z^0_{tag_{18},NS} \end{pmatrix} \end{pmatrix}$$

  where NS is equal to $NbSamplings$ and $(x^0_{tag_1,i}, y^0_{tag_1,i}, z^0_{tag_1,i})$ are the positions of the tag 1 at the sampling i in the model reference frame.

- `TUTILISES` : it is a boolean matrix of 18 rows and $NbSamplings$ columns. The (i, j) component is at $False$ if the tag i is occuluded at the j sampling and at $True$ otherwise.

Then the call to the **ConvertFromCsvToTags** function is:

```
[TOBS, TUTILISES] = ConvertFromCsvToTags(filename);
```

Four step are implemented in the **ConvertFromCsvToTags** function :

1. First step: the tags are reorganized and the occluded tags are identified;
2. Second step: the occluded tags are interpolated;
3. Third step: a frame change (from Optotrak frame to model reference frame) is computed;
4. Fourth step: the data are filtered.

The following sections explain these different steps.

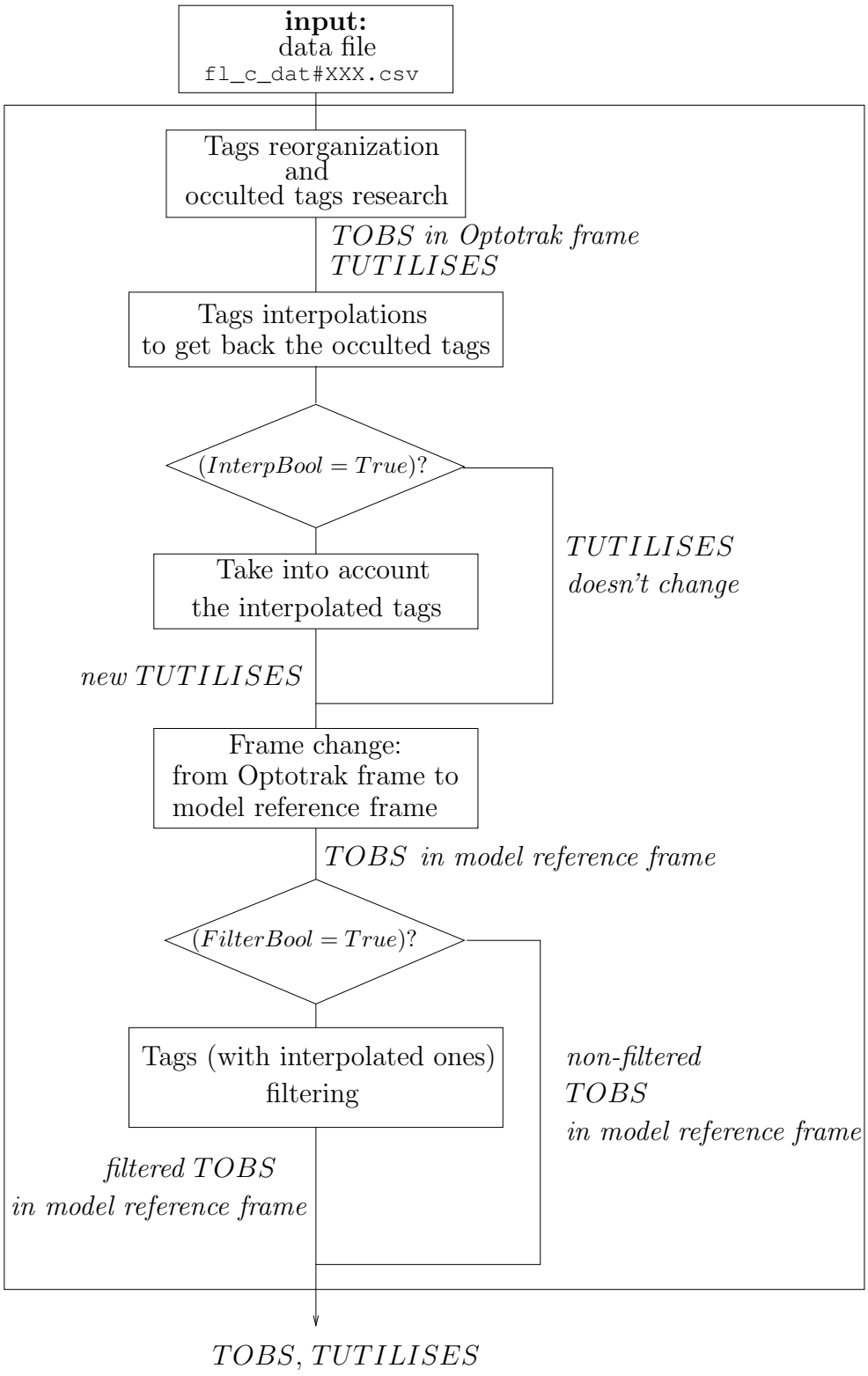


Figure 9.12: Global view of the **ConvertFromCsvToTags** function

First step: the reorganization of the tags and the identification of the occluded tags

The number of the leds used in experiments is greater than the number of tags used in the reconstruction. In the reconstruction, the tags used (if there is no occlusion) are the tags 1 to 18. Then, the first step is to choose the tags corresponding to the model ones and to put their positions in the right order in a matrix of 18 row and $NbSamplings$ columns. These positions are given in the Optotrak frame.

Then, the occluded tags are identified. The function computes the `TUTILISES` matrix which is a boolean matrix of 18 rows and $NbSamplings$ columns.

Second Step: signal interpolation to get back the occluded tags positions

The Optotrak sets the tag coordinates to a given value(-3.6973E+028) if this tag is occluded. Then these coordinates are replaced by the interpolated ones. The interpolation is made in two steps:

1. We consider separately the coordinates x, y and z of one tag along the time. For each coordinate of each tag, the scilab function **splin** is called. This function computes a cubic spline S which interpolates the (t_i, x_i) points (in the case of x component), where t_i are the times samplings for which the considered tag is not occluded;

2. Then the scilab function **interp** values $x(t_i) = S(t_i)$ where t_i are the time samplings for which the considered tag is occluded.

Then the user can choose to take into account the interpolated values or not. A boolean `InterpBool` allows him to make this choice. If `InterpBool` is set to $True$, the `TUTILISES` variable will be modified and the interpolated tags will be taken into account. If it is set to $False$, the `TUTILISES` variable will not be modified and the interpolated tags would be considered as occluded in the reconstruction. By default, `InterpBool` is set to $True$.

Third step: frame change

The observed tags positions must be given in the model reference frame. Then a frame change is made to put in the `TOBS` matrix the observed tags positions expressed in the model reference frame. In experiments, there is only one fixed led. Then, we considered that the led fixed on the right and left malleolus and on the great trochanter did not move during the first 20 samplings. We furthermore assumed that the middles of the malleolus leds and of the great trochanter leds were placed on the y axes (vertical) of the model reference frame origin.

Fourth step: data filtering

The observed tag positions can be filtered. The filter is a Butterworth low-pass filter. By default, the order of the filter is 2 the sampling frequency is 100 Hz and the cut-off frequency is 8 Hz. The user can change the order, the sampling frequency or the cut-off one by modifying respectively the `p`, `SamplingFrequency` and `CutOffFrequency` variables in the **ConvertFromCsvToTags** function.

The scilab function used for the filtering are the **iir** (to compute the filter) and the **flts** (to apply the filter) functions.

The `FilterBool` boolean allows the user to filter or not the observed tags positions. If it is set to $True$, the filter is apply. If it is set to $False$, the filter is not apply.

9.6.5 The Reconstruction.sci function

The `Reconstruction` function allows us to reconstruct a position from tags positions given by optical sensors. The input data are:

- `Tobs` : The `Tobs` parameter is a matrix of $NumberOfTags$ rows and 3 columns. The row i is the observed (by optical sensors as Optotrak) position (x, y, z) of the tag i. These positions are given in the sensor frame.
- `Tutilises` : The `Tutilises` parameter is a column boolean vector of size $NumberOfTags$. Its i-th component is set to $False$ if the tag i was occluded and $True$ otherwise.
- `q_init` : The `q_init` parameter is the initial position in the minimisation algorithm.
- `nb_iter` : The `nb_iter` parameter is the maximum number of iteration in the minimisation algorithm.
- `seuil` : the `seuil` parameter is the threshold in the minimisation algorithm. When the error between the reconstructed tags and the observed tags is smaller than `seuil`, the minimisation algorithm stops and the corresponding position $qrec$ is returned.

Then, the call to the **Reconstruction** function is:

```
> [qrec, info] = Reconstruction(q_init, Tobs, Tutilises, nb_iter, seuil);
```

where `qrec` is the reconstructed position and `info` an integer informing on the potential error that could have appear in the **Reconstruction** function. This error is the same one returned by the algorithm used for the minimisation which is the **fsqp** algorithm[3].

[3]the **fsqp** algorithm was developped by the AEMDESIGN: http://www.aemdesign.com/

9.6.6 The AllSubjectsTrajectoriesReconstruction.sci script

This script allows the user to reconstruct all the trajectories of all the subjects given by the user in the **ReconstructionParameters.csv** file. For each subject XX recorded in the **ReconstructionParameters.csv** file, this script calls the StaticTrajectoriesReconstruction.sci script with the subject parameters recorded in this parameter file and for all the trajectories stored in the `OpticalSensors/JXC(+ or -)P(+ or -)/SubjectXX`.

9.6.7 Utility functions

The visualization functions

Checking of the reconstructed position: the VisuTagsComparison.sci function The function **VisuTagsComparison** allows us to see the result of the reconstruction and the measured tags on the same graphic. The call to this function is of the form:

```
> VisuTagsComparison(QREC, TOBS, TUTILISES);
```

where:

- `QREC` is the matrix of the reconstructed trajectory. This matrix has 33 rows and $NbSamplings$ columns;
- `TOBS` is the matrix of the observed tags. Its size is $NumberOfTAgs \times (3 * NbSamplings)$ (see section 9.6.4);
- `TUTILISES` is the boolean matrix which specifies the occluded tags and the non-occluded tags (see section 9.6.4).

The WriteInCsvFile.sci and WriteInFile.sci functions

The WriteInCsvFile.sci function The **WriteInCsvFile.sci** function allows the user to store matrix data in `.csv` file. The call to this function is of the form:

```
> WriteInCsvFile(filename, M, HText, VText);
```

where `filename` is the name of the output file, `M` is the matrix data of n row and m columns, `HText` and `VText` are row vectors of respectively m and $(n+1)$ rows. Then the output file written of this form:

$$\begin{array}{ccccc} Vtext(1), & HText(1), & HText(2), & \dots & HText(m) \\ VText(2), & M11, & M12, & \dots & M1m \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ VText(n+1), & Mn1, & Mn2, & \dots & Mnm \end{array}$$

with columns separated by commas and lines by RC.

The WriteInFile.sci function The **WriteInFile.sci** function returns an output file which is in the following format:

$$\begin{array}{cccccc} sample & Text(1) & Text(2) & \dots & Text(m) \\ 1 & M11 & M12 & \dots & M1m \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ n & Mn1 & Mn2 & \dots & Mnm \end{array}$$

with columns separated by tabs and lines by RC.

The call to the **WriteInFile.sci** function is of the form:

```
> WriteInFile(filename, M, Text);
```

where `filename` is the name of the output file, `M` is the matrix data and `Text` the text written in the first row of the output file.

Bibliography

[1] H-anim. http://h-anim.org/.

[2] *Biomechanics and Motor Control of Human Movement*. Second edition, 1990.

[3] *Anatomie 1, l'appareil locomoteur*. 1992.

[4] C. E. Clauser, J. T. McConvillet, and J. M. Young. Weight, volume, and center of mass of segments of the human body. Technical Report AMRL-TR-69-70, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Dayton, OHIO, 1969.

[5] Laboratoire d'Anthropologie Appliquee. Ergodata: Dictionnaire des mesures. http://www.biomedicale.univ-paris5.fr/LAA/FR/index.htm.

[6] P. De Leva. Adjustments to zatsiorsky-seluyanov's segment inertia parameters. *Journal of Biomechanics*, 29(9):1223–1230, 1996.

[7] Pr.NGUYEN HUU. Anatomic schemas. http://www.univ-brest.fr/S_Commun/Biblio/ANATOMIE/Web_anat/index.htm.

[8] Craig T. Lawrence and André L. Tits. A computationally efficient feasible sequential quadratic programming algorithm. *SIAM J. on Optimization*, 11(4):1092–1118, 2000.

[9] G. Wu, S. Siegler, P. Allard, C. Kirtley, A. Leardini, D. Rosenbaum, M. Whittle, D.D. D'Lima, L. Cristofolini, H. Witte, O. Schmid, and I. Stokes. Isb recommendation on definitions of joint coordinate system of various joints for the reporting of human joint motion-part i: ankle, hip, and spine. *Journal of Biomechanics*, 35(4):543–548, 2002.

[10] G. Wu, FC. van der Helm, H.E. Veeger, M. Makhsous, P. Van Roy, C. Anglin, J. Nagels, A.R. Karduna, K. McQuade, X. Wang, F.W. Werner, and B. Buchholz. Isb recommendation on definitions of joint coordinate systems of various joints for the reporting of human joint motion–part ii: shoulder, elbow, wrist and hand. *Journal of Biomechanics*, 38(5):981–992, 2005.

[11] V. M. Zatsiorsky, L. M. Raitsin, V. N. Seluyanov, A. S. Aruin, and B. J. Prilutzky. *Biomechanics and Performance in Sport*, chapter Biomechanical characteristics of the human body, pages 71–83. Bundeninstitut für Sportwissenschaft, Germany, 1993.

[12] V. M. Zatsiorsky and V. N. Seluyanov. *Biomechanics VIII-B*, chapter The mass and inertia characteristics of the main segments of the human body, pages 1152–1159. Human Kinetic, Illinois, 1983.

[13] V. M. Zatsiorsky, V. N. Seluyanov, and L. G. Chugunova. *Contemporary Problems of Biomechanics*, chapter Methods of determining mass-inertial characteristics of human body segments, pages 272–291. CRC Press, Massachusetts, 1990a.

[14] V. M. Zatsiorsky, V. N. Seluyanov, and L. G. Chugunova. *Biomechanics of Human Movement: Applications in Rehabilitation, Sports and Ergonomics*, chapter In vivo body segment inertial parameters determination using a gamma-scanner method, pages 186–202. Bertec, Ohio, 1990b.