Muscle activation Database

An Experimentation Project

Name: Cindy Müllenmeister
Student number: 3722252
Teacher: Nicolas Pronost
Date: January 6, 2012

Table of Contents

1. Introduction

In the last few years motion capturing is used more often to create realistic looking animation. One example of the use of motion capturing the movie Avatar. In this movie James Cameron used motion capturing with real actors (see [1]) for the animation of the aliens and non-human creatures (see [2]). Other examples can be found in the gaming industry where animation by using motion capturing occurs more and more. The game series of Assassin's Creed is an excellent example (see [3]).
The most often used optimization techniques are applied in the area of films, games and computer graphics and are, as Komura et al. (2000) mentioned, purely for the "purpose of animation" [4]. Optimization techniques interpolate or retarget the motion capturing data to a given model so that the model will move according to the movements of the motion capturing. The models of the human body used for purpose of animation are "simple multibody without any physiological structure" (see [4]). In other words, the anatomical structure of the human body is neglected.
In the medical field and the field of biomechanics , researchers take motion capturing one step further with methods that take the anatomical structure into consideration like Komura et al. (2000) used the motion capturing in their research (see [4]). Despite the possibilities of further advancing the animation with anatomical structures, the developed techniques in the medical fields and the field of biomechanics are not used for the purpose of animation. Instead, the researchers are more concerned with trying "to understand the underlying dynamics of the movement of living beings, from gait research" [4] and to find and treat patients with for instance gait problems (see [4]) with help of motion capturing and animation.

Komura et al. (2000) mentioned [4] that the anatomical body is important for the characteristics of postures and motions, even at relaxed postures. So the muscles in our bodies have influence on the behavior of joints and torques of the models. Introducing anatomical structure to the models could improve the realism of the animation that results from the motion capturing. In 1996, Parke et al. (1996) already tried to improve the realism of facial animation with the use of taking the muscles movements into account (see [10]). At that time, the computer power was too little to implement the method designed by them. Nowadays, the power of the computer is greater than years ago. Therefore, the next step of the animation would be to take this anatomical structure into account just like in the field of biomechanics.
As the research of Komura et al. (2000) shows, the creation of animation with anatomical structure is time consuming. To speed up the creation of the animation based on motion capturing data, a look-up table would be one possibility, with other words, a database that would contain at least muscle activation. Therefore, a first step into the direction of using anatomical structure in the animation for the animation community would be the creation of a database of muscle excitation.

The creation of such a database is described later on in this paper. It consists of two parts. The first part is computing the muscle activation from a motion capturing file. This is done in a so called pipeline. The pipeline receives a motion captured file (c3d file) as input and follows specific steps to construct muscle excitation as output. Some of these steps are not performed in real-time, therefore the pipeline will not run in real-time. The second part is the creation of the database itself as a module of RAGE. RAGE is a library that consists of many different modules. These modules are used for analyzing and working with animation data such as motion capturing. The module is designed to perform online and uses steps that are not time consuming like the pipeline. For this reason the aim is to get the database run online and as fast as possible.
This paper starts with the description of these specific steps, followed by the description of the pipeline itself. Next, the programming of the RAGE module is described. In the last part of the paper some experimenting with the database and the pipeline is described.

2. Muscle activation

Muscle activation is also known as muscle excitation. The calculation of the muscle excitation is according to Delp et al. (2007) still a major challenge [9]. One recent technique to calculate it is the Computer Muscle Control (CMC), which still costs such an amount of computation time that real-time processing is still not possible (see [9]). Before the CMC can be calculated, 3 other steps have to be performed: Scaling, Inverse Kinematics and Residual Reduction Algorithm. The input needed for these steps are: a musculoskeletal model, experimental kinematics and reaction forces and moments (see [9]). The experimental kinematics is actually the motion capturing data such as the coordinates of the markers. The reaction force and the moments are measured with so called force plates. These force plates are placed and used during the motion capturing and calculates the forces and moments for every frame in the x-, y- and z-direction (see [6]). The relation between the individual steps can be found in figure 1.

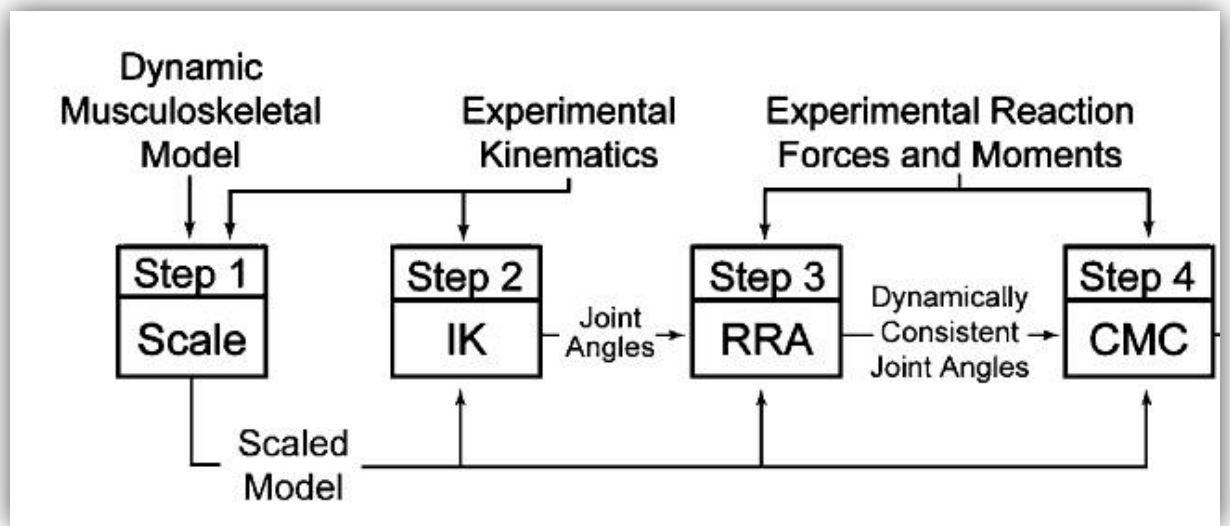

Figure 1: Relation between the steps ([9], p.1943)

2.1 Scaling

The first step is scaling of the model so that it matches the "anthropometry of an individual subject" ([9], p. 1943). In the scaling process mass properties, muscle fiber length and dimension of each body segment and other properties of the model are scaled (see [6], p.21). This is done with the distance between markers of the motion captured data in relation to the distance between virtual markers of the model. A second step in the scaling is the use of inverse kinematics. The inverse kinematics "adjust the locations of the virtual markers" ([6], p.21) in such a way that they better matched with the location of the markers of the motion captured data. Scaling errors can be reduced this way.

2.2 Inverse Kinematics (IK)

Inverse Kinematic is a problem that needs to be solved to receive joint angles and translations that fit the marker data of the motion capturing best (see [9]). Solving of this problem is done in the second step. Furthermore, the IK problem is known as a least-squares problem. It takes the difference between the marker location of the motion captured data and the location of virtual markers of the model and minimizes this difference, subject to joint constraints" [9].

2.3 Residual Reduction Algorithm (RRA)

Experimental errors and modeling assumptions cause often dynamically inconsistence between the model kinematics received from the second step and the ground reaction forces and moments (see [9], p. 1943). The third step, the Residual Reduction Algorithm is an algorithm with which this inconsistence will be helped or at least reduced (see [9], p.1943). The algorithm consists of two passes (see [6]). The first pass is the altering of the mass "so that excessive leaning in the left-right or fore-aft directions is corrected" [6].The second pass consists of altering "the kinematics of the model to be more consistent with the ground reaction data" [6]. For this, three residual forces and three residual moments are applied to the segments of the model "to control the six degrees of freedom between the model" [9] and the world (three rotations and three translations).

2.4 Computer Muscle Control (CMC)

The last of the four steps is the Computer Muscle Control (CMC). The CMC is used to calculate the muscle excitations (see [9]) that (as Delp et al mention) "produce a coordinated muscle-driven simulation of the subject's movement". The set of muscle excitation is computed in such a way, that it is able to "produce a coordinated muscle-driven simulation of the subject's movement" ([9], p. 1944).

To compute the muscle excitation, the CMC uses 5 steps. The first step is the solving of the initial state of the model (see [6]). After that, the CMC computes "a set of desired accelerations" ([6], p. 22). For this, it makes use of the proportional-derivative (PD) controller (see [6]). Next, the CMC tries to minimize the "error between the model coordinates and the inverse kinematic coordinates" ([6], p.22) by applying the following equation:

$$\vec{k}_v = 2 \cdot \sqrt{\vec{k}_p}$$ (see [6])

Finally, the CMC computes the activation of the muscles and the muscle actuators with help of the static optimization technique.

The output of the CMC is a file containing the muscle activations, the muscle length and joint angles per frame of the motion capturing.

The whole process is not possible to perform in real-time as mentioned before. For example, Delp et al. (2007) mentioned that the CMC "that reproduce measured pedaling dynamics" took 10 minutes to compute it (see [9]).

3. Pipeline

The pipeline makes use of the Matlab Toolbox from A. Sandholm [6] and the program OpenSim as described below. Because the toolbox is written in Matlab, it was decided to program the pipeline in Matlab.

3.1 OpenSim

OpenSim is a program that enables the user to model, simulate and analyze musculoskeletal systems [9]. The first introduction of the program was at the American Society of Biomechanics Conference in 2007 and was used by thousands of people since then (see [7]).
OpenSim contains a variety of features including scaling, Inverse Kinematics (IK), Residual Reduction Algorithm (RRA) and Computer Muscle Control (CMC).
The program can be used over its interface which is written in Java (see [7]). First, before working with this interface, the user has to check the default properties of it.
Through the interface, the commands for the Scaling, IK, RRA and CMC can be carried out with hand or with *configuration* files. To be able to use these commands in the pipeline, they need to be accessible outside of the interface, which they are. Using them outside the interface requires the use of *configuration* files. Therefore, these files need to be generated before using the OpenSim commands. The Matlab toolbox is able of generating these *configuration* files automatically.

3.2 Matlab Toolbox

The Matlab Toolbox used in the pipeline was presented by Sandholm et al [6]. The toolbox contains features like reading the c3d files, translating data into model coordination system, applying noise reduction, creating motion files with a specific format and creating *configuration* files used in OpenSim ([6], pp.38-39).
All those features are used in our pipeline. The first time using the Matlab Toolbox, it seemed that the function *loadLabels* was missing and also *session_setup* as mentioned in [5] p. 40. Therefore, a new function *loadLabels* was created during the process of programming the pipeline as described below. Furthermore, the function *writeXML* had to be changed for the pipeline (see 3.2.2).
The *configuration* files make use of additional files containing additional (partly user depended) information, for example the *configuration* file for Scaling uses the additional file *Scale_IK_Tasks.* These additional files need also be created with the Matlab Toolbox. Unfortunately, functions that enable the user of the Matlab toolbox to automatically create these additional files are missing completely. Therefore, new functions that will be capable of creating the additional files need to be implemented.

3.2.1 LoadLabels

According to the code of the toolbox, the function *loadLabels* gets a *c3dkey* as a parameter and returns this parameter at the end of it. The code of the toolbox further indicated that the toolbox adds a structure named *glab* to the *c3dKey* which contains joint names, muscle names, FP suffix and prefix and a few other variables/structures. As it was not clear in the coding what exactly those names, prefix, suffix and other variables/structures should contain, an older version of the toolbox was used as guidance. The older version of the toolbox contains *loadLabels*, but as a script and not as a function. This script contains the following items:

1a) Force Plate Naming Convention
1b) Coordinate Vector Setup
1c) Vicon / OpenSim Offset Marker
1d) Joint / Muscle Model

2a) Force Plate Labels
2b) Force Plate Directions
2c) GRF Labels
2d) CoP Labels
2e) GRM Labels (Origin)
2f) GRM Labels (CoP)
3a) joint labels
3b) muscle labels
4) EMG labels
5) Markerset labels

The toolbox already contains 5 scripts for the joint and muscle labels: *loadJointDelp23*, *loadJointDelp31*, *muscleDelp92*, *loadMuscleAnderson54* and *loadMuscleAnderson54Patella*. To make *loadLabels* more flexible and easy to switch names, used joint scripts, used muscle scripts and values, all the items of the script *loadLabels* (except 3a and 3b as the scripts for those already existed) were divided into the scripts *loadFPLabDependancy*, *loadFPPlotting*, *loadEMGLabels* and *loadMarkerLabels*. The function *loadLabels* then only opens all the scripts and assigns everything in the structure *glab* to the *c3dkey*. At the moment, the *loadLabels* loads the scripts *loadJointDelp23* for the joints and *muscleDelp92* for the muscles and can be changed any time to other muscle and joints sets as needed. The full code for the function *loadLabels* and the additional scripts can be found in Appendix A.

3.2.2 WriteXML changes

The *writeXML* function in the toolbox is the one that creates the *configuration* files for the Scaling, IK, RRA and CMC commands of OpenSim. To be correctly used by the pipeline, a few changes had to be made. The first change was the adding of an extra parameter for a filtering frequency. In the *configuration* file for RRA and CMC, a filter frequency is defined. Before the change, *writeXML* would always set the value of the frequency to -1 because it was hard coded to do so. Now, the user can define the filter frequency beforehand.

The next change that was made in the function *writeXML* was the order of all the lines of the *configuration* files for Scaling, RRA and CMC. This was to make it easier in comparing it to example files to see what lines and values were still missing.
For writing the *configuration* file for the scaling, the height, age and note were taken out as they never change of values and are of no use for the scaling itself. Furthermore, the function for writing the scaling *configuration* file always checks if manual scaling was used or not. As it is planned to always use an extra file containing the Scale set, this check was not necessary anymore.

Another major change made in this function is for the creation of the *configuration* file for CMC. Here, the function always assumed a RRA will be done before the CMC and therefore creates a *configuration* file for the CMC that will use the result of the RRA. It could happen that the RRA was not necessary in which case the CMC *configuration* file has to use the IK data's instead of the RRA. Therefore, the major change here is a check whether or not RRA had been done before. If RRA had been done, nothing changes. If not, the CMC *configuration* file uses now the IK data's instead of the RRA.

3.2.3 Additional Functions

The *configuration* file for the scaling, IK, RRA and CMC created by the toolbox all need extra files to run correctly in OpenSim. The *configuration* file for the Scaling needs 3 different extra files, the *configuration* file for the IK needs only 1 extra file, the *configuration* file for the RRA and the

configuration file for the CMC need 3 extra files. This makes in total 10 extra files needed for the whole process.
The Matlab Toolbox has no function that is able to create these files automatically. This is why extra functions are necessary that are able to create these files automatically. As most of the content of the 10 files are different, 10 different functions were added to the toolbox, one for every file. These functions are: *writecmcActuators*, *writecmcconstrain*, *writeCMCTask*, *writeIKTask*, *writerraactuators*, *writerraconstrain*, *writeRRATask*, *writeScaleMeasuremenSet*, *writeScaleSet* and *writeScaletask*.
The content of all these files can be split into two categories: user depended content and non-changeable content. The non-changeable content stays the same with every motion. Therefore, it was decided to hard code it into the functions. The user depended content on the other hand is different. This content can be different from motion to motion and the number of it can also vary. Furthermore, it is not possible to predict the content beforehand. This means that the user has the responsibility to give the functions these contents. This user depended content has therefore to be placed into xml file which all the functions will read out and pick their part for their files from.
So, the 10 functions work all the same: first open the file which contains the user depended content, search for the part needed in that function, creating an xml file with the user depended content and the hard coded content.

Additional to the extra files, the *configuration* files require a *trc*-file of the motion. A *trc*-file is a file that contains the time and the coordinates (x,y and z coordinates) of every marker for every frame in a specific format. It is possible to create this trc-file beforehand with a program called *Mokka*. This program is able to save the c3d information in a trc-file. Nevertheless, the aim is that the pipeline will do every step automatically on its own and this includes also the creation of the *trc*-file. Unfortunately, the command to save the c3d file as a trc-file from the program *Mokka* is not accessible from the command line. This also means that the pipeline cannot access it. Because all the information needed is already accessible thanks to the Matlab Toolbox, the best way to get a *trc*-file is a new function that saves one itself with all the information it gets from the Matlab Toolbox. The function for this is called *generateTRCFile*. This function takes the *c3dKey* as a parameter which is created with the Matlab Toolbox beforehand. It then takes all the information needed for a *trc* file out of the *c3dKey* and prints it in a new created *trc*-file according to the specific format of the *trc*-file.

With those additional functions, the Matlab Toolbox is complete for the pipeline and creates everything needed for the pipeline to run smoothly.

3.3 Setup for the pipeline

Before the user can run the pipeline, he/she has to set everything up in the way the pipeline will expect it.

First of all, the input of the pipeline will be *c3d* files. The amount of *c3d* files can be different every time the pipeline will be run. So that the pipeline will know the number of motions and the name of each one of them, the user has to fill specific information into a file called *motions*. The file *motions* was first a txt-file where only the names of all motions had to be written in. During the first design of the database, it became clearer that additional information for every motion for the database was required, such as the sort of motion.
Therefore, the file was changed to an xml file. The file *motions.xml* contains one section called *motions*. This section contains as many subsections as there are motions for the pipeline with the name *mot*. Every *mot* section has exactly 4 subsections: *name*, *group*, *subgroup* and *person*. The section *name* contains the name of the motion, whereby the section *person* contains the name of the person who performed the motion. As the design of the database also requires the knowledge of the sort of motion that is performed, the section *group* was added that will contain this information. Furthermore, some motions can have additional information such as degree of the feet position

during a motion of the sort crouch. This extra information can be written in the section *subgroup*. It is however possible that for some motions no information (that could be fit into for example the subgroup) exists or the user does not want to specify this. In that case it was first tried to leave that section empty. The Matlab *xml_parser* then automatically filled in the information of the previous section. Therefore, it was decided that the user needs to fill that section with a dot instead.
So, the file *motions.xml* should for example for two motions look like this:

```
<motions>
        <mot>
                <name>Gait_0004_label</name>
                <group>gait</group>
                <subgroup>.</subgroup>
                <person>Tobias</person>
        </mot>
        <mot>
                <name>Gait_0007_label</name>
                <group>gait</group>
                <subgroup>Testsubgroup</subgroup>
                <person>Matthijs</person>
        </mot>
</motions>
```

Next to this file, the user has to create a folder named *Common*. This folder should contain all model files that will be used in the scaling and all the reference files for all the motions. What happens with these files will be explained in chapter 3.4.

The next preparation the user has to make before starting the pipeline is creating folders for every c3d file with the same name as the c3d file next to the folder *Common*. In those folders, the user needs to place the c3d files and an xml file called *setup*. You can find an example of the resulting file hierarchy in figure x.

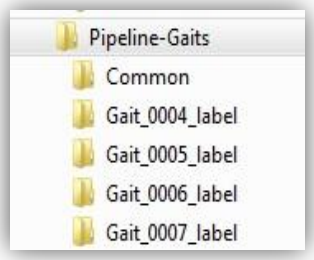

Figure 2a: Folder Hierarchy

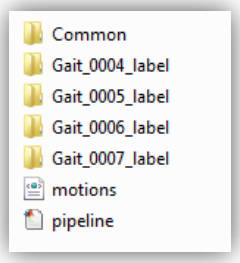

Figure 2b: Content of main folder Pipeline-Gait

Figure 2c: Content of one motion folder (Gait_0004_label)

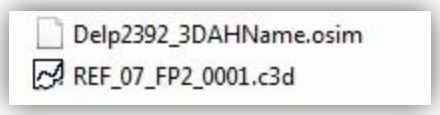

Figure 2d: Content of the folder Common

The file *setup.xml* contains 4 sections next to the head section: *Scale*, *IK*, *RRA* and *CMC*.
In the other four sections, the user can first specify if the pipeline should run the OpenSim commands with the same names under the section *use*. If the user decides to not use one of these commands, he/she has to make sure that the folder contains a file of the same sort and name as the one this command would produce, otherwise the pipeline will fail.
In the section *scaling*, the user needs to specify the name of the reference file in the section *motionFile* and the model that is scaled according to the reference file in the section model. Next to these sections, the user also needs to specify *ScaleTasks* with name and weight. These *ScaleTasks* are the user depended content of the IK Task that the *configuration* file for the scaling needs (see 3.2.3).
In the section IK, the user only needs to specify *IKTasks* with name and weight for the additional Task of the IK *configuration* file.

Section *RRA* needs to be filled with the subsection *filter_frequency* containing the filtering frequency for the *configuration* file and subsections *Actuators* with name and optimal force, which are needed to create the Actuator file for the RRA *configuration* file. The number of subsection *Actuators* depends on the user itself. Next to these sections, the section *RRA* also contains a section called *Task*. In this section the user needs to specify joints with name, kp, kv and weight of it. This user specified information will be used for the generation of the Task file for the RRA *configuration* file.

The sections *CMC* contains the same subsections as the section RRA with the difference that the sections *joint* only contain name, kp and kv. It is possible that the user fills the subsections with exactly the same values as the subsections of the section *RRA*. Because it is possible that the user will use different values in the CMC than in the RRA, it was decided to use two different sections for the CMC and for the RRA. An example of this *configuration* file for the motion Gait_004_label can be found in Appendix B.

3.4 Walking through the pipeline

The pipeline starts with reading in the file *motions.xml*. The pipeline takes the name of the first motion and enters the folder of the same name. In that folder, it will open the *setup.xml* file. The first section of this file is *Scaling*. If the user specified the use of it as true, the pipeline will look into the folder *Common* if a folder exists with the same name as specified in section *motionFile*. If the folder exists, the pipeline will just copy the scaled model out of this folder into the motion folder and is done with the scaling. If the folder does not exist, the pipeline will create it, copy the reference motion file and the model file into it, create the *configuration* file for the scaling and all the needed extra files and use the scaling command of OpenSim to scale the model with the reference file. The resulting scaled model is then copied to the motion folder.
Next in the *configuration* file is the IK. If this is set on true, the pipeline will create the IK *configuration* file and the additional files that the *configuration* file requires. After creating these files, the pipeline will run the IK command of OpenSim.
If the user specified the use of the RRA as true, the pipeline will then create the *configuration* file for OpenSim and the additional needed files. After that, the pipeline will run the RRA with OpenSim. For the RRA, OpenSim does not have a separate command. Instead, the command cmc.exe is used with the RRA *configuration* file. The content of the RRA *configuration* file makes sure that a RRA is then performed instead of a CMC.
CMC is now the next section. The pipeline will again make sure if the user stated the use of it as true. If not, it will just skip it. If it is stated as true, the pipeline will create the *configuration* file for the CMC. For this, it has to check whether or not the user used the RRA. When the RRA was not used, the *configuration* file will only make use of the results from the IK. Otherwise, it will make use of results of the RRA. Furthermore, the additional files are created and the pipeline runs at the end the CMC command with OpenSim.
The result of the CMC is a file containing the muscle activation of the (first) motion. This file is the result of the pipeline for the motion and is therefore the output of the pipeline. Now, the pipeline could go further with the next motion in the motions.xml file, but before it does so, the pipeline needs to do something else: combining the data of different files into one for the database. Most of the data the pipeline needs to put into the file for the database are already loaded into it: the four sections (group, subgroup, person and name of the motion) from the *motions.xml* file and data from the model. The only data that the pipeline additional needs for the file for the database are the muscle activations from the results of the CMC. To read the file that contains the muscle activation, a new function was created: *readCMC*. It reads the whole file that contains the muscle activation, calculates the minimum and maximum value of each joint angle and muscle activation and gives it to the pipeline. The pipeline saves the data of a motion in an xml file called *databaseQuery* under a new section called mot.

The pipeline is now finished with processing of a motion of the file *motions.xml*. It repeats these steps for every other motion in the file. After the last motion was processed, the pipeline will print on the command line that it is finished.

The whole pipeline was programmed in two files. The first file is called *pipeline* and is a Matlab script. This script reads the *motions.xml* file, the scaling and the preparations for the pipeline. The second file, *runPipeline*, is a function. It gets information from the first script and does all the other steps: IK, RRA and CMC.

4. Rage Module

The library RAGE, RAGE stands for "Real-time AGS Game Engine" [8], consists of many different modules. The modules are for analyzing and working with motion capturing, animation platform and others. It is possible to extend the program with new modules. The languages used for the program are C++ and Python. C++ is used for the calculations and process running (in the background) such as calculating physics for example. Python is used for the interface and communication with the graphic engine Ogre and RAGE library.

4.1 The database design

The muscle activation database is programmed as a new module for RAGE. The module should enable the user to search for motions in the database and that (another) module is then opened with the results of this search.

The first question for this is: On what should the user be able to search? All the results from the scaling can be used as parameters and search criteria's. The same goes for the results of the IK, RRA and CMC. Also, additional information received from the user can also be integrated into the database and used as search parameters. To use all of these parameters/information as search parameters would be a lot to program and also would give the user too many choices. Therefore, it is decided to limit it to the results of the CMC (to be more precise: the joint angles and the muscle activation), body sets of the models from the scaling and the additional data from the user. It is possible to change these later to other search criteria.

The next question that needs to be answered is: How should the user chose criteria? The first thought was to put the possible values of the search criteria in multi-selectable lists. This would mean that the user would have many lists with values out of which he/she could choose one or more values. In the case of this database, the user would have 11 different lists to choose from: *Names*, *Scaled models*, *Groups*, *Subgroups*, *Persons*, *Duration of the CMC*, *Bodysets*, *Mass*, *Center of Mass*, *CMC joints and CMC muscles.*

The lists for the mass of the body set, for the muscle activation and for the joint angle would only contain numbers. The number of values in these lists can be very large and the difference between the numbers in the lists can be very small. Therefore, it can become hard for the user to use the lists with these numbers and could also become time consuming with searching for all these different values. A better solution for these search criteria's is to let the user enter minimum and maximum values. The database then has to search the motions where the values of the mass, joint angles and/or muscle activation lies between the given minimum and maximum.

So far, the design of the database shows how the user can search through the motions and on what data the database is able to look for. The only question that remains now is: How to load in the database? It is possible that the user used the pipeline for more than one run in different folders, but wants to combine these motions now in the database. Furthermore, the pipeline is already programmed in such a way, that it also produces a xml file containing all information the database needs as mentioned before. Therefore, the decision is to let the user browse to those xml files and read in the information contained in these files and build the database based on them. As users are, he/she might make a mistake, for example load accidentally in more than he/she actually wanted. To help solve this mistake, the database will have an additional button: clear. With this button, the user can unload the whole database and restart with the loading of it.

4.2 Programming the Interface of the Database

The interface of the database is programmed in Python and can be found in the script *muscleDatabasetest.py*. This script contains two classes: *databaseGUI* and *resultGUI*.

4.2.1 Class databaseGUI

The class *databaseGUI* is the class that is called upon from the main part of the script. First, it creates a new module. This module contains a file browser and two buttons next to it (Create Database button and a Clear button), 8 empty lists and 8 text fields where the user can enter minimum and maximum values. Another button called Search motions is under these lists and text fields. You can see the entire interface layout in figure 3.

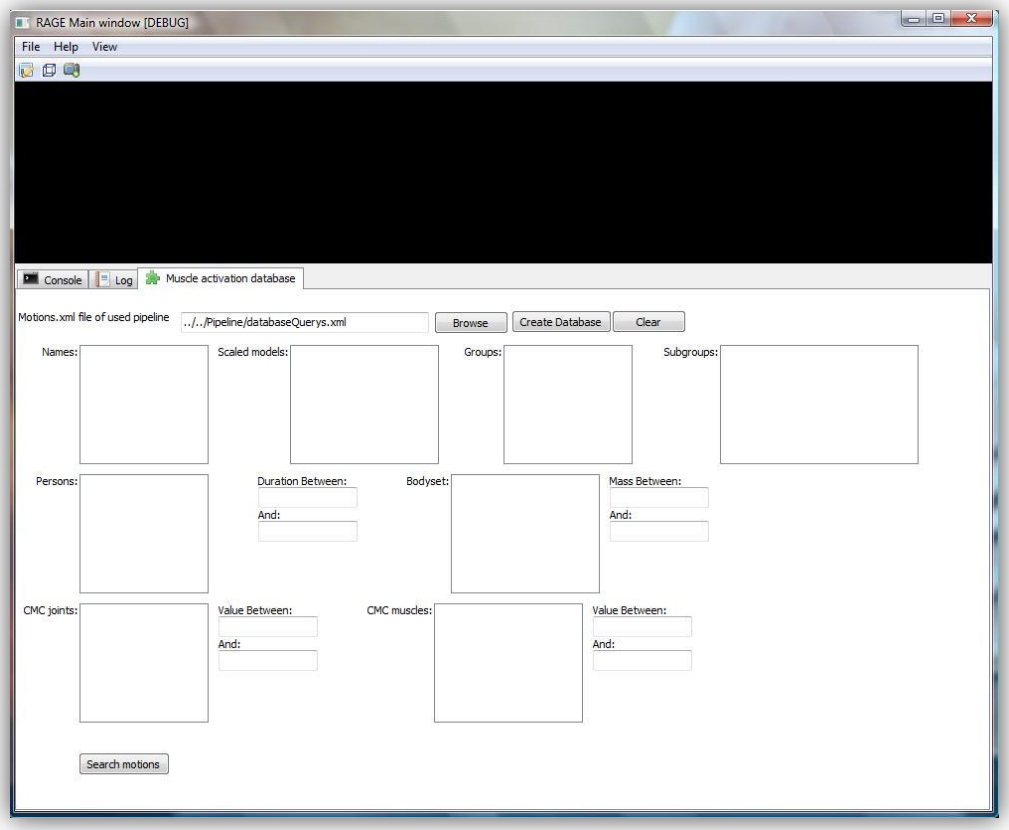

Figure 3: Interface of the Database query

The file browser from the wx.lib needs a function that is called as soon as one file is chosen. Because the user could chose a wrong file, the function for this, *ApplyMapFile*, is a dummy function that just prints the name of the chosen file.

The button Create Database makes use of the function *CreateDatabase* as soon as it is selected. *CreateDatabase* is a function that reads every motion from the file the user chooses with the file browser and uses the functions *givenMotionData*, *handleModelData* and *handleCMC* to fill the list and handing down the information to the C++ part.

The function *givenMotionData* reads the group, subgroup, motion name, model name and person names, fills different lists with these values and hands the information down to the C++ part.

The function *handleModelData* is responsible to read in for the body set data's, fills lists with it and hands the data down to the C++ part. *handleCMC* does the same with the CMC related part of the xml file.

So after pressing the button Create Database, the lists will be filled (see figure 4).

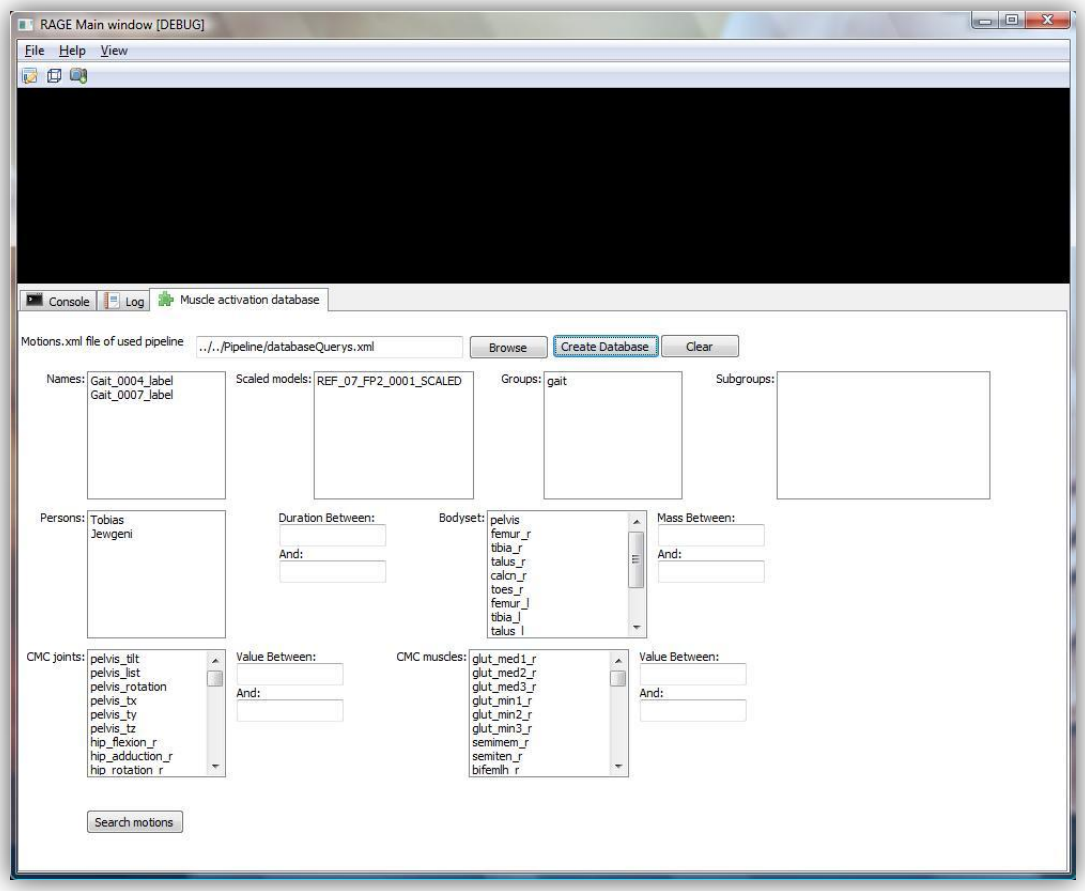

Figure 4: Database filled with data

The button Clear has the opposite effect. It will clear all the lists with the function *ClearDatabase*. Next to clearing all the lists the user can see, it also hands down the command of clearing everything to the C++ part.
Search motions is bound to the function *SearchMot*, this function calls functions for every list and text field to get the user input of it and hands it down to the C++ part. Finally, it calls the function *getResult*. This function calls the search function of the C++ part, saves the result in the global variable results and creates a new instance of the class *resultGUI*.

4.2.2 Class resultGUI

The class *resultGUI* is actually a simple one. It is creating a new module in the existing interface and fills the module with all motion names found in the global results (see figure 5).

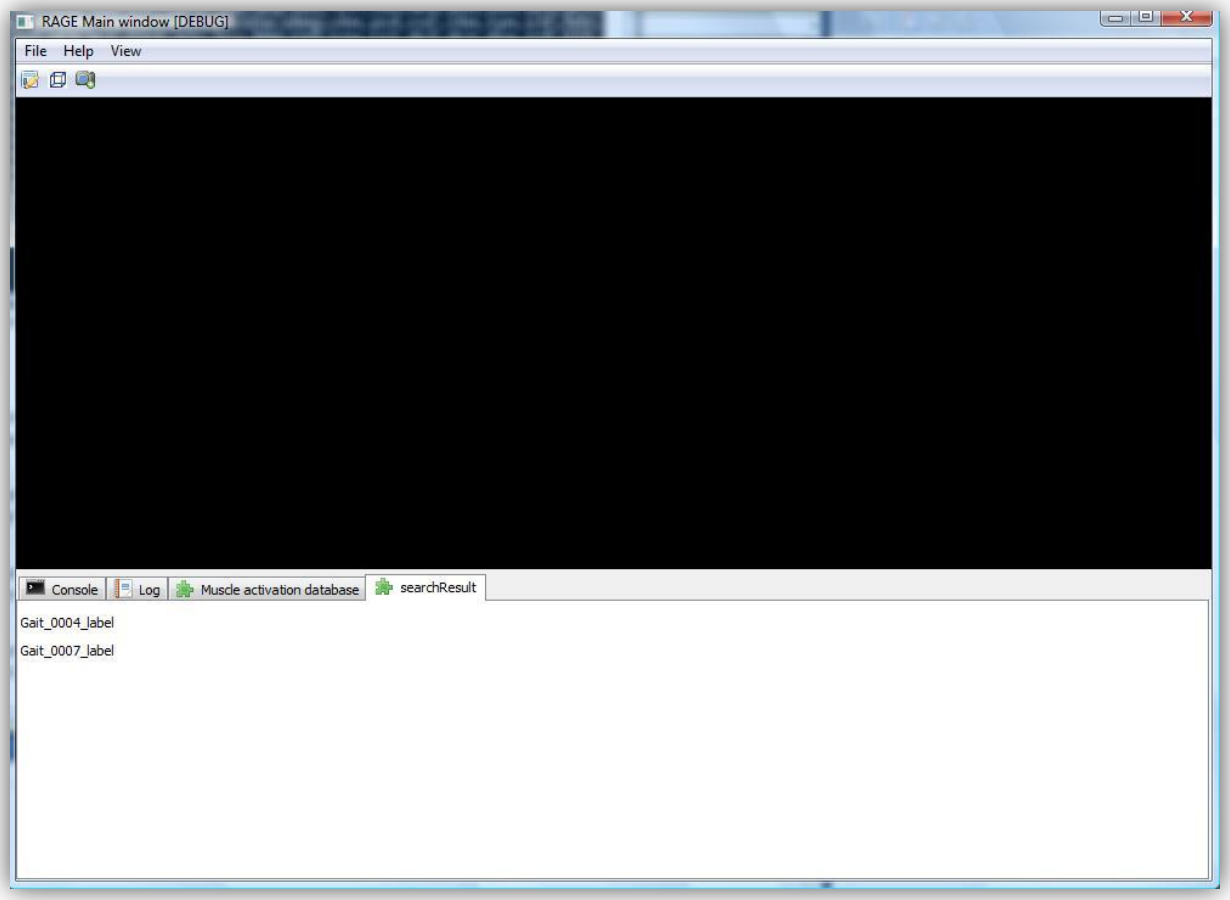

Figure 5: Tab with the result of the search

4.3 Programming the main part of the database

The main part of the database cannot be found in the interface but in the part that runs in the background, the C++ part. This is the part that actually keeps track of the every motion with all its information in the database. The whole C++ code consists of 4 Objects/classes: *Body*, *Model*, *MotionInf* and *SearchEngine*.

4.3.1 Class Body

The class *Body* is meant to keep track of all the values of the body sets. Therefore, it has variables for the mass, the inertia, the center of mass and the name of the body set. Other classes can get these variables with the different get-functions of this class.

4.3.2 Class Model

Because a model can have more than one body set, this class was designed to create one object for every model with a name and all its body sets. The body sets can be added with the set function and retrieved with the get function. Because there can be more than one body, the get function needs the name of the body set to retrieve the right one as a parameter. The name is set as soon as the constructor is called. Therefore, it has no set function, only a get function to retrieve the name.

4.3.3 Class MotionInf

MotionInf contains all the information belonging to one motion. This includes one object of the model belonging to the motion, the name of the motion, the group and subgroup it belongs to, the person performing the motion, the duration of the CMC and two containers, to be more specific maps. A map is container from the standard library in C++ that stores the elements formed by a combination of keys and mapped values. One contains the minimum and maximum value of the joint angles map with the joint names, and one contains the minimum and maximum values of the muscle activation mapped with the muscle name as key. All the variables are set in the constructor (except the two maps) and have therefore no set function. The class does however contain a get function to retrieve the values in the variables. For the two maps, there exist set functions. These functions get the names, the minimum value and the maximum value and insert it into the maps. The get functions for these maps retrieve of the minimum and maximum values of the map according to the name (the mapping key) given in the parameter.

4.3.4 Class SearchEngine

The class *SearchEngine* is actually the main class. It creates the *MotionInf* and Model objects if they don't exist yet, maps them with their names as keys and keeps this way track of them. The objects of the models are mapped extra with their name as key despite the fact that they are already mapped in *MotionInf.* The reason for this is that more than one motion can contain the same model. If a new motion is now inserted into the *SearchEngine* with a model that already exist in the database, the object of the model can easily be found in the map and linked to the motion. Having two objects containing the exact same information would be a waste of space.
Beside these maps, the class has also 9 vectors. These vectors are filled with all the search parameters, which the user selected in the interface, with set functions. The function *clearSearch* empties these vectors. The vectors containing the search parameters have to be emptied after every search. If the user clicks the button Clear in the interface, the whole database here has to be cleared. This is what the function *clearDatabase* does: it empties the two maps which in fact are the whole database.
The most important function in this class is the *searchMotions* function. This function goes through the map of motions. For every motion, the function looks at the information of the motion starting with the name and compares it with the values in the search vector that belongs to that information. For example looking at the name of the motion and comparing it with the values of the vector *searchName*. If one of the values in the vector is the same or the vector is empty and this is the same for every other search vectors, the name of the motion is added to a resulting vector. This resulting vector is returned at the end of the function.

5. Experiments

The design of the database is a first setup and can be changed later as wished. During the design of the database and the pipeline, there was for some points more than one option in how to program/build it. For these points and to see if the goal of a database works as fast as possible and in real-time, experiments were conducted to see what the best solution would be.

5.1 Objects vs. mapping

The first idea during the designing of the database was to use mappings instead of objects. For every kind of information, one map would be used that maps strings of the information to a vector containing motion names that have these information, for example a map for the groups in which the string *Gait* maps to a vector containing the names *Gait_0004* and *Gait_0007*. Other information part that would contain extra information, for example body sets containing mass, center of mass

and inertias of the body set, would not map a vector to string, but instead a structure. The structure would then contain the extra information and a vector with the motions belonging to that information. A mapping of the body sets would then map a structure to the name of the body and the structure would contain the mass, center of mass and inertia next to the vector of motions, as example.

The database would now need n such maps. These n maps contain m keys with vectors and the vectors contain k values. The search for motions in the worst case scenario (with l search criteria's for every information) would then need a time of O((n*m*l)+(s*k)) where s is the number of vectors that match the search criteria's.

Another idea was to use objects instead of mapping. An object of the motion would then contain m information. In a search (with l search criteria's for every information) through n motions would in the worst case scenario need a time of O(n*m*l). This is slightly better. Furthermore, this approach is more object orientated than the approach with the mapping and will therefore result in better maintenance possibilities and extending possibilities future. Because of that, objects are used instead of mapping in the database.

Table 1: Worst case calculations

Number of motions	Number of different kind of information	Number of different values for the informations	Number of different search criteria's for every information	Number of found Vectors	Length of vectors	O((n*m*l)+(s*k))	O(n*m*l).
10	20	10	10	10	1	O((20*10*10)+(10*1)) = O(2010)	O(10*20*10) =O(2000)
20	20	20	20	20	1	O((20*20*20)+(20*1)) = O(8020)	O(20*20*20) =O(8000)
20	20	20	10	10	1	O((20*20*10)+(10*1)) = O(4010)	O(20*20*10) =O(4000)
30	20	30	30	30	1	O((20*30*30)+(30*1)) = O(18030)	O(30*20*30) =O(18000)
30	20	30	10	10	1	O((20*30*10)+(10*1)) = O(6010)	O(30*20*10) =O(6000)
50	20	50	50	50	1	O((20*50*50)+(50*1)) = O(50050)	O(50*20*50) =O(50000)
50	20	50	10	10	1	O((20*50*10)+(10*1)) = O(10010)	O(50*20*10) =O(10000)
100	20	100	100	100	1	O((20*100*1100)+(100*1)) = O(200100)	O(100*20*100) =O(200000)

5.2 Reading CMC vs. Min and Max

As was described in the design of the database, the user can search with minimum and maximum values muscles activations and/or joint angles. For this search, two different ways were thought about.
The first way is that the pipeline calculates the minimum and maximum values beforehand and that the database only needs to look up if the minimum value given by the user falls between the minimum and maximum value of the calculated one.
The other ways of implement the search was that the database itself has to look into the CMC file and that the pipeline gives only the directory in which the file can be found.

To be able to compare these two ways, both were implemented and run. Let's first look at the pipeline by letting it run twice with two motions. The first run is without calculating the minimum and maximum values of the joint angles and muscle activity and the second is run with these calculations. In the setup of the two motions the IK is defined as not to use and for one motion is even all the scaling, RRA and CMC set to not be used.
The first run of the pipeline took 9 minutes and 15 second until it was finished. The second run on the other hand took even longer until it was finished: 11 minutes 57 seconds. So the second run took 2 minutes 42 minutes longer than the first run. Because the files the pipeline opens and uses to calculate the minimum and maximum are the same size of both motions, the time it takes should be equal for both motions. This means that reading the file and calculating the minimum and maximum for every joint angle and muscle activation takes for every motion 1 minute 21 seconds. Because both motions have 185 frames, this would be 0.478 seconds per frame.

The run of the pipeline with and without calculation of the min and max values was also done once with three motions and once with six motions. In all the motions are the use of the Scaling, IK, RRA and CMC set to false. These were calculated beforehand with the pipeline because only the time it needed for the calculation of the min and max values are of interest here. You can find the total time the pipeline needed to and the calculated time difference between the run with the calculation of the minimum and maximum values and without in table 2.

Table 2: Time the pipeline took to run with different amount of motions and the calculated difference between running with and without calculating min and max values

Number of motions	With calculation of min and max values?	Total Time of the pipeline	Time difference	Time difference per motion	Time difference per frame
2	No	9 min 15 sec	2 min 42 sec	1 min 21 sec	0.478 sec
2	Yes	11 min 57 sec			
3	No	6 min 36 sec	3 min 23 sec	1 min 7.7 sec	0.367 sec
3	Yes	9 min 59 sec			
6	No	13 min 01 sec	7 min 00 sec	1 min 10 sec	0.378 sec
6	Yes	20 min 01 sec			

Now, let's look at the database. First, the database was loaded in twice: once with the minimum and maximum values and once without the minimum and maximum values. Both loading processes seemed not to make any time difference. It is possible that with an amount of hundred of motions that the difference between these loading processes would be measurable, but it probably would make only a slightly difference.

When the minimum and maximum values are not loaded in, the database still needs to be able to search through the values of the joint angles and/or muscle activation to see if it contains values that are between the maximum and minimum value of the user. For this, another function was implemented: *readCMCFile*. This function reads a CMC file in and scans through the lines. For every line, it looks for the *i*-th column. The number (*i*), for the number of the column, has to be given to the function through the parameter. For now, this function is only a test function and reads one hard coded file and can later be changed to a variable file-address. Another button (Test reading File) was added to the interface that would call this function with the number 3 (see figure 6).

Figure 6: the new test button in the RAGE module

Because the first time clicking on this button it was not clear when the function finished reading through the file, a breakpoint was placed at the end of the new implemented function. Now clicking on the button until the break point was reached took between 3 and 4 seconds. Just like the motions in the pipeline, consist the file the database reads of 185 frames. This means that the database takes between 0.016 seconds and 0.022 seconds for the reading of one frame. On the other hand, clicking on the search button (which still uses the calculated minimum and maximum values of the joint angles and muscle activity as described in chapter 4.3) finished in less than 1 second. The difference between 3 and 1 second does not seem much, but the test was only conducted with one motion. Searching through files for 100 motions would then take approximately 300 seconds, while on the other hand searching with the calculated minimum and maximum values for 100 motions would take approximately 100 seconds or even less.

The experiments with the pipeline and the database show that the pipeline will consume more time for calculating the minimum and maximum values than it actually saves for the database search. The time, the pipeline needs to calculate the min and max values, is (as the experiments show in table 2) almost totally proportional. Proportionally does every frame of all the motions consume a time of approximately 0.478 seconds for the computation of the min and max values. In comparison to the time the database needs to read the one frame of the CMC file by itself is it a lot of time. On the other hand, the pipeline already needs a lot of time to perform the steps necessary for the calculation of the muscle activation as you can see in table 2. The database only takes one or even less than one second to start up and is able to run online.
Furthermore, reading the CMC file in the database gives the opportunity to even look at the values at every frame. This would make it possible to search for frames in the CMC that are for example between a specific minimum and maximum and return those frames next to the name of the motion as result of the search.
Because the aim is to make the database as fast as possible, it was chosen to calculate the minimum and maximum values and use them in the database search despite the longer time it takes in the pipeline. Furthermore, it is enough for now to only return the name of the motions that fit the search criteria instead of including the frame numbers in the result. It is possible to change this if it is

required because the function that reads the file already exists. The only changes that have to be made is making the reading more flexible (for example by letting the pipeline give the folder and file name) and take the part of calculating the min and max values in the pipeline out of it. Next to this, the returned and in the new tab printed result has also be changed to giving and printing the found frame numbers.

6. Conclusion

In this paper, the design, implementation and experimenting of a muscle activation database was described. This muscle activation database tries to make a step of the motion capturing for animation purpose in the direction of taking the anatomical structure into consideration as it is already done in the field of Biomechanics. The next steps would be to use this database to further Nevertheless, the time the database will need will rise with more motions in the database which can result in the inability to run online. Because the aim was that the database will run online and as fast as possible, it was decided to let the pipeline calculate the min and max values of the muscle activations and joint angles.

The search criteria's and input of these can still be changed if wished. If other search criteria's or other way of searching is chosen, it is recommended to test if the database will still run with these changes in real-time as this is the aim of the database.

References

[1] The Art of Motion Capturing in Avatar, Retrieved December 28, 2011 from http://www.vizworld.com/2009/12/art-motion-capture-avatar/

[2] Avatarblog, Behind the Scenes Look at the Motion Capture Technology Used in AVATAR, Retrieved December 28, 2011 from http://avatarblog.typepad.com/avatar-blog/2010/05/behind-the-scenes-look-at-the-motion-capture-technology-used-in-avatar.html

[3] Assassin's Creed 2 – Motion Capture, Retrieved December 28, 2011 from http://assassinscreed.uk.ubi.com/community-news/?p=3391

[4] T. Komura, Y. Shinagawa and T.L. Kunii, "Creating and retargeting motion by the musculoskeletal human body model" in *The Visual Computer*, vol. 16, 2000, pp.254-270

[5] A. Sandholm, "Neuromuscular Modeling and Visualization of the Human Lower Extremities" at the *ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE*, Phd Thesis nr. 4923, 2011

[6] A. Sandhol, N. Pronost and D. Thalmann, "MotionLab: A Matlab toolbox for extracting and processing experimental motion capture data for neuromuscular simulations" in *Proceedings of the Second 3D Physiological Human Workshop (3DPH) 2009, Lecture Notes in Computer Science*, vol. 5903, 2009, pp. 110-124

[7] F. Anderson, E. Guendelman , A. Habib, S. Hamner, K. Holzbaur, C. John, J. Ku, M. Liu, P. Loan, J. Reinbolt, A. Seth and S. Delp, "OpenSim User's Guide" at *https://simtk.org/home/opensim*, Release 2.2, September 13, 2010

[8] What does RAGE stand for?, Game technology lab, Retrieved December 28, 2011 from http://www.gametechnology.nl/index.php?option=com_content&view=article&id=49:rage-name&catid=27:rage&Itemid=54

[9] S. L. Delp, F. C. Anderson, A. S. Arnold, P. Loan, A. Habib, C. T. John, E. Guendelman and D. G. Thelen, "OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement" in IEEE *Transactions on Biomedical Engineering*, vol. 54, nr. 11, November 2007, pp. 1940 – 1950

[10] F. I Parke and K. Waters, "Computer Facial Animation", 1996, ISBN 1-56881-014-8

Appendix A

Function loadLabels:

```
function C3Dkey = loadLabels(C3Dkey)
loadFPLabDependancy;
loadFPPlotting;
loadJointDelp23;
muscleDelp92;
loadEMGLabels;
loadMarkerLabels;
glab.modelJointData = glab.jointDelp23;         % Joint convension
glab.muscleModel = glab.muscleDelp92;        % Muscle convension
C3Dkey.glab = glab;
```

Script loadFPLabDependancy:

```
% ================================
% LAB DEPENDANCY
% ================================
% Ia) Force Plate Naming Convention
% ---------------------------------------------------------------------
% Force plate numbers MUST be set up within the lab.
% The names of the channels of these force plates are defined here.
%       EXAMPLE
%       glab.FP.string = '%s%d%s';
%       glab.FP.prefix  = {'FP','FP','FP','FP','FP','FP'};
%       glab.FP.suffix  = {'Fx','Fy','Fz','Mx','My','Mz'};
%
%       Moment about FP #3 in the X direction analog channel = 'FP3Mx'
%       Please note that the glab.FP.string IS CASE SENSITIVE
% ---------------------------------------------------------------------
glab.FP.string = '%s%d%s';      % Prefix, PlateNum, Suffix
glab.FP.prefix  = {'FP','FP','FP','FP','FP','FP'};
glab.FP.suffix  = {'Fx','Fy','Fz','Mx','My','Mz'};
glab.FP.verticalForceIndex = 3;

% Ib) Coordinate Vector Setup
% ---------------------------------------------------------------------
% This section sets transformation vectors that are crucial to the
% rigid body transformations needed between the Force Plate, Vicon,
% and the Model coordinate systems.
%
% Example: FP coord system -> Model coord system
%         glab.dirVec.FPMODEL(3,:)  = [2 -3 -1];
%
%         MODEL X = FP Y
%         MODEL Y = FP -Z
%         MODEL Z = FP -X
% ---------------------------------------------------------------------
% X direction (Vicon) Gait -> transformation vectors
glab.dirVec.FPMODEL(1,:)  = [-1 -3 -2];   % FP coord system    -> Model coord system
glab.dirVec.VICMODEL(1,:) = [1 3 -2];     % Vicon coord system -> Model coord system

% -X direction (Vicon) Gait -> transformation vectors
glab.dirVec.FPMODEL(2,:)  = [1 -3 2];   % FP coord system    -> Model coord system
glab.dirVec.VICMODEL(2,:) = [-1 3 2];   % Vicon coord system -> Model coord system

% Y direction (Vicon) Gait -> transformation vectors
```

```
glab.dirVec.FPMODEL(3,:)  = [2 -3 -1];   % FP coord system    -> Model coord system
glab.dirVec.VICMODEL(3,:) = [2 3 1];     % Vicon coord system -> Model coord system

% -Y direction (Vicon) Gait -> transformation vectors
glab.dirVec.FPMODEL(4,:)  = [2 -3 -1];   % FP coord system    -> Model coord system
glab.dirVec.VICMODEL(4,:) = [-2 3 -1];   % Vicon coord system -> Model coord system


% 1c) Vicon / OpenSim Offset Marker
% ---------------------------------------------------------------------
% This section sets marker to be used as the offset point to
% line up the model to the walking platform in OpenSim
% ---------------------------------------------------------------------
glab.offsetMarker = 'SCM';
```

Script loadFPPlotting:

```
% ================================
% FORCE PLATE PLOTTING
% (Used by getKinetics.m)
% ================================

% 2a) Force Plate Labels
% ---------------------------------------------------------------------
glab.name{1} = 'GRF';
glab.name{2} = 'CoP';
glab.name{3} = 'GRMo';
glab.name{4} = 'GRMx';

% Force Plate Directions
% (x dir+-, y dir+-, z dir+-)
% ---------------------------------------------------------------------
glab.dir{1} = {'Fore',     'Aft'};              % X
glab.dir{2} = {'Vertical', ''};                 % Y
glab.dir{3} = {'Lateral',  'Medial'};           % Z

% GRF Labels
% (label, units, directionSign+, directionSign-)
% ---------------------------------------------------------------------
glab.S{1} = [{'GRF X (right)', 'N'}, glab.dir{1}(1), glab.dir{1}(2)];
glab.S{2} = [{'GRF Y (right)', 'N'}, glab.dir{2}(1), glab.dir{2}(2)];
glab.S{3} = [{'GRF Z (right)', 'N'}, glab.dir{3}(1), glab.dir{3}(2)];
glab.S{4} = [{'GRF X (left)',  'N'}, glab.dir{1}(1), glab.dir{1}(2)];
glab.S{5} = [{'GRF Y (left)',  'N'}, glab.dir{2}(1), glab.dir{2}(2)];
glab.S{6} = [{'GRF Z (left)',  'N'}, glab.dir{3}(1), glab.dir{3}(2)];

% CoP Labels
% (label, units, directionSign+, directionSign-)
% ---------------------------------------------------------------------
glab.X{1} = [{'CoP X (right)', 'm'}, glab.dir{1}(1), glab.dir{1}(2)];
glab.X{2} = [{'CoP Y (right)', 'm'}, glab.dir{2}(1), glab.dir{2}(2)];
glab.X{3} = [{'CoP Z (right)', 'm'}, glab.dir{3}(1), glab.dir{3}(2)];
glab.X{4} = [{'CoP X (left)',  'm'}, glab.dir{1}(1), glab.dir{1}(2)];
glab.X{5} = [{'CoP Y (left)',  'm'}, glab.dir{2}(1), glab.dir{2}(2)];
glab.X{6} = [{'CoP Z (left)',  'm'}, glab.dir{3}(1), glab.dir{3}(2)];

% GRM Labels (Origin)
% (label, units, directionSign+, directionSign-)
% ---------------------------------------------------------------------
glab.Mo{1} = [{'GRMo X (right)', 'Nm'}, glab.dir{1}(1), glab.dir{1}(2)];
glab.Mo{2} = [{'GRMo Y (right)', 'Nm'}, glab.dir{2}(1), glab.dir{2}(2)];
glab.Mo{3} = [{'GRMo Z (right)', 'Nm'}, glab.dir{3}(1), glab.dir{3}(2)];
glab.Mo{4} = [{'GRMo X (left)',  'Nm'}, glab.dir{1}(1), glab.dir{1}(2)];
glab.Mo{5} = [{'GRMo Y (left)',  'Nm'}, glab.dir{2}(1), glab.dir{2}(2)];
```

```
glab.Mo{6} = [{'GRMo Z (left)',  'Nm'}, glab.dir{3}(1), glab.dir{3}(2)];

% GRM Labels (CoP)
% (label, units, directionSign+, directionSign-)
% ------------------------------------------------------------------------
glab.Mx{1} = [{'GRMx X (right)', 'Nm'}, glab.dir{1}(1), glab.dir{1}(2)];
glab.Mx{2} = [{'GRMx Y (right)', 'Nm'}, glab.dir{2}(1), glab.dir{2}(2)];
glab.Mx{3} = [{'GRMx Z (right)', 'Nm'}, glab.dir{3}(1), glab.dir{3}(2)];
glab.Mx{4} = [{'GRMx X (left)',  'Nm'}, glab.dir{1}(1), glab.dir{1}(2)];
glab.Mx{5} = [{'GRMx Y (left)',  'Nm'}, glab.dir{2}(1), glab.dir{2}(2)];
glab.Mx{6} = [{'GRMx Z (left)',  'Nm'}, glab.dir{3}(1), glab.dir{3}(2)];
```

Script loadEMGLabels:

```
% ================================
% EMG LABELS
% ================================
% (label, name)
% Used by getEMG.m to know which EMG analog channels to extract
% from the C3D file
% ------------------------------------------------------------------------

% Full Biotel EMG set
% -------------------
glab.BiotelEMG{1}  = {'VM'   ,'Vastus Medialis'};
glab.BiotelEMG{2}  = {'VL'   ,'Vastus Lateralis'};
glab.BiotelEMG{3}  = {'RF'   ,'Rectus Femoris'};
glab.BiotelEMG{4}  = {'GMAX' ,'Gluteus Maximus'};
glab.BiotelEMG{5}  = {'GM'   ,'Gluteus Medius'};
glab.BiotelEMG{6}  = {'BF'   ,'BF'};
glab.BiotelEMG{7}  = {'SOL'  ,'Soleus Lateral'};
glab.BiotelEMG{8}  = {'TA'   ,'Tibialis Anterior'};

% Test EMG set
% --------------------
glab.testEMG{1}  = {'VM'   ,'Vastus Medialis'};
glab.testEMG{2}  = {'GM'   ,'Gluteus Medius'};
glab.testEMG{3}  = {'SOL'    ,'Soleus'};
glab.testEMG{4}  = {'TA'   ,'Tibialis Anterior'};
```

Script loadMarkerLabels:

```
% ================================
% MARKERSET LABELS
% ================================
% Used by getMarkers.m to know which markers to extract
% from the C3D file
% ------------------------------------------------------------------------

glab.markersStatic = {'RTPL', 'LTPL', 'SFS', 'TRX', 'STM', 'TI', 'RAM', 'LAM', 'RCIL', 'LCIL', 'RASIS', 'LASIS', 'RPSIS', 'RPSIS', 'SCM', 'RGT', 'LGT', 'RMFE', 'LMFE', 'RLFE',
'LLFE', 'RTT', 'LTT', 'RFH', 'LFH', 'RMEDMAL', 'LMEDMAL', 'RLATMAL', 'LLATMAL', 'RDISTI', 'LDISTI', 'RDIST5', 'LDIST5', 'RBIGTOE', 'LBIGTOE', 'RHEEL', 'LHEEL', 'RThT', 'RThB',
'RThR', 'RThF', 'LThT', 'LThB', 'LThR', 'LThF', 'RShT', 'RShB', 'RShR', 'RShF', 'LShT', 'LShB', 'LShR', 'LShF'};

glab.markersDynamic = {'RTPL', 'LTPL', 'SFS', 'TRX', 'STM', 'TI', 'RAM', 'LAM', 'RCIL', 'LCIL', 'RASIS', 'LASIS', 'RPSIS', 'RPSIS', 'SCM', 'RGT', 'LGT', 'RMFE', 'LMFE', 'RLFE',
'LLFE', 'RTT', 'LTT', 'RFH', 'LFH', 'RMEDMAL', 'LMEDMAL', 'RLATMAL', 'LLATMAL', 'RDISTI', 'LDISTI', 'RDIST5', 'LDIST5', 'RBIGTOE', 'LBIGTOE', 'RHEEL', 'LHEEL', 'RThT', 'RThB',
'RThR', 'RThF', 'LThT', 'LThB', 'LThR', 'LThF', 'RShT', 'RShB', 'RShR', 'RShF', 'LShT', 'LShB', 'LShR', 'LShF'};
```

```
%glab.markers = {'RTPL', 'LTPL', 'SFS', 'TRX', 'STM', 'TI', 'RAM', 'LAM', 'RCIL', 'LCIL', 'RASIS', 'LASIS', 'RPSIS', 'RPSIS', 'SCM', 'RGT', 'LGT', 'RMFE', 'LMFE', 'RLFE', 'LLFE',
'RTT', 'LTT', 'RFH', 'LFH', 'RMEDMAL', 'LMEDMAL', 'RLATMAL', 'LLATMAL', 'RDISTI', 'LDISTI', 'RDIST5', 'LDIST5', 'RBIGTOE', 'LBIGTOE', 'RHEEL', 'LHEEL', 'RThT', 'RThB',
'RThR', 'RThF', 'LThT', 'LThB', 'LThR', 'LThF', 'RShT', 'RShB', 'RShR', 'RShF', 'LShT', 'LShB', 'LShR', 'LShF'};

glab.markers(1).name = 'RTPL';
glab.markers(2).name = 'LTPL';
glab.markers(3).name = 'SFS';
glab.markers(4).name = 'TRX';
glab.markers(5).name = 'STM';
glab.markers(6).name = 'TI';
glab.markers(7).name = 'RAM';
glab.markers(8).name = 'LAM';
glab.markers(9).name = 'RCIL';
glab.markers(10).name = 'LCIL';
glab.markers(11).name = 'RASIS';
glab.markers(12).name = 'LASIS';
glab.markers(13).name = 'RPSIS';
glab.markers(14).name = 'LPSIS';
glab.markers(15).name = 'SCM';
glab.markers(16).name = 'RGT';
glab.markers(17).name = 'LGT';
glab.markers(18).name = 'RMFE';
glab.markers(19).name = 'LMFE';
glab.markers(20).name = 'RLFE';
glab.markers(21).name = 'LLFE';
glab.markers(22).name = 'RTT';
glab.markers(23).name = 'LTT';
glab.markers(24).name = 'RFH';
glab.markers(25).name = 'LFH';
glab.markers(26).name = 'RMEDMAL';
glab.markers(27).name = 'LMEDMAL';
glab.markers(28).name = 'RLATMAL';
glab.markers(29).name = 'LLATMAL';
glab.markers(30).name = 'RDISTI';
glab.markers(31).name = 'LDISTI';
glab.markers(32).name = 'RDIST5';
glab.markers(33).name = 'LDIST5';
glab.markers(34).name = 'RBIGTOE';
glab.markers(35).name = 'LBIGTOE';
glab.markers(36).name = 'RHEEL';
glab.markers(37).name = 'LHEEL';
glab.markers(38).name = 'RThT';
glab.markers(39).name = 'RThB';
glab.markers(40).name = 'RThR';
glab.markers(41).name = 'RThF';
glab.markers(42).name = 'LThT';
glab.markers(43).name = 'LThB';
glab.markers(44).name = 'LThR';
glab.markers(45).name = 'LThF';
glab.markers(46).name = 'RShT';
glab.markers(47).name = 'RShB';
glab.markers(48).name = 'RShR';
glab.markers(49).name = 'RShF';
glab.markers(50).name = 'LShT';
glab.markers(51).name = 'LShB';
glab.markers(52).name = 'LShR';
glab.markers(53).name = 'LShF';
```

Appendix B

The file setup.xml for the motion Gait_0004_label:

```
<Setup xml_tb_version="3.1">
 <Scale>
            <use>false</use>
  <motionFile>REF_07_FP2_0001</motionFile>
  <model>Delp2392_3DAHName</model>
  <ScaleTask name="RAM">
   <weigth>5</weigth>
  </ScaleTask>
  <ScaleTask name="LAM">
   <weigth>5</weigth>
  </ScaleTask>
  <ScaleTask name="RASIS">
   <weigth>150</weigth>
  </ScaleTask>
     [...]
  <ScaleTask name="LBIGTOE">
   <weigth>50</weigth>
  </ScaleTask>
 </Scale>
 <IK>
            <use>false</use>
  <IKTask name="RAM">
   <weigth>10</weigth>
  </IKTask>
  <IKTask name="LAM">
   <weigth>10</weigth>
  </IKTask>
     [...]
  <IKTask name="LDIST5">
   <weigth>10</weigth>
  </IKTask>
  <IKTask name="LBIGTOE">
   <weigth>20</weigth>
  </IKTask>
 </IK>
 <RRA>
  <use>false</use>
  <filter_frequency>6</filter_frequency>
  <Actuators name="hip_flexion_r">
   <optimal_Force>300.0</optimal_Force>
  </Actuators>
  <Actuators name="hip_adduction_r">
   <optimal_Force>200.0</optimal_Force>
  </Actuators>
       [...]
  <Actuators name="lumbar_rotation">
   <optimal_Force>200.0</optimal_Force>
  </Actuators>
  <Task>
   <rdCMC_Joint name="pelvis_tz">
    <kp>100.0</kp>
    <kv>20.0</kv>
    <weigth>5.0e0</weigth>
   </rdCMC_Joint>
   <rdCMC_Joint name="pelvis_tx">
    <kp>100.0</kp>
    <kv>20.0</kv>
    <weigth>5.0e0</weigth>
```

```
    </rdCMC_Joint>
          [...]
    <rdCMC_Joint name="lumbar_rotation">
      <kp>100.0</kp>
      <kv>20.0</kv>
      <weigth>1.0e1</weigth>
    </rdCMC_Joint>
  </Task>
 </RRA>
 <CMC>
                <use>false</use>
  <filter_frequency>6</filter_frequency>
  <Actuators name="hip_flexion_r">
    <optimal_Force>1.0</optimal_Force>
  </Actuators>
  <Actuators name="hip_adduction_r">
    <optimal_Force>1.0</optimal_Force>
  </Actuators>
        [...]
  <Actuators name="lumbar_rotation">
    <optimal_Force>1.0</optimal_Force>
  </Actuators>
  <Task>
    <rdCMC_Joint name="pelvis_tz">
      <kp>100.0</kp>
      <kv>20.0</kv>
    </rdCMC_Joint>
    <rdCMC_Joint name="pelvis_tx">
      <kp>100.0</kp>
      <kv>20.0</kv>
    </rdCMC_Joint>
          [...]
    <rdCMC_Joint name="lumbar_rotation">
      <kp>100.0</kp>
      <kv>20.0</kv>
    </rdCMC_Joint>
  </Task>
 </CMC>
</Setup>
```

APPENDIX C

Experimenting with user dependent content of the IK

After the pipeline was implemented and tested successfully, it was used to get the muscle activation of the motion Gait_0004. The user depended content for this run can be found in Appendix B. The scaling and the IK seemed to have run successfully with it, but the steps RRA and CMC seemed to fail. The RRA and the CMC uses both the resulting file of the IK. Therefore, the first thing to look at was the resulting motion of the IK using the interface of OpenSim. The interface of OpenSim allows the user to load a model (in this case the model that results from the scaling) and motions, like the motions created by the IK, and displays it. As you can see in figure 7, the hip of the skeleton model is twisted.

Figure 7: resulting motion of the IK twisted

The explanation for this is that the weight user dependent content used in the IK is not correct for the markers at the hip. Therefore, it was experimented with the user depended content with the aim of removing the twist in the hips.

In the experiment, the weights of markers were changed, then the pipeline ran again with these changed weights and look was taken at the resulting IK motions, to see if it changed and the twist disappeared. Because only the hip is twisted, it is reasonable to only experiment with the weight of the markers around the hip. Therefore, only the weight of the markers RASIS, LASIS, SCM, RCIL and LCIL changed during these experiments.

In the first experiment, the weights were raised to 30. The resulting IK motion did not look any different than the one before. Therefore, the change in weights did not change the IK enough.
Next, the weights of the other markers were lowered to see if this makes any difference. The weight of 10 was changed to a weight of 1 and the weight of 20 was changed to a weight of 2 of all markers (except of the markers around the hips, which remained 30). Running the pipeline with these changes also did not seem to effect the twist at all as it still looks the same.

The thought that rose was that the raise of the weights maybe was not enough. Therefore, the weights of the markers around the hip were raised to 1000. The weights of the other markers

remained like they were in the experiment before (around 1 and 2). You can find all the weights with marker weight to which it belongs in table 3.

Table 3: Marker names and their weight for the IK

Marker name	Weight
RAM	1
LAM	1
STM	1
RASIS	1000
LASIS	1000
SCM	1000
LPSIS	1000
RCIL	1000
LCIL	1000
RLFE	1
RMFE	1
RGT	1
RThT	2
RThB	2
RThF	2
RThR	2
RLATMAL	1
RMEDMAL	1
RTT	5
RFH	5
RShT	2
RShB	2
RShF	2
RShR	2
RHEEL	2
RDIST1	1
RDIST5	1
RBIGTOE	2
LLFE	1

LMFE	1
LGT	15
LThT	2
LThB	2
LThF	2
LThR	2
LLATMAL	1
LMEDMAL	1
LTT	5
LFH	5
LShT	2
LShB	2
LShF	2
LShR	2
LHEEL	2
LDIST1	1
LDIST5	1
LBIGTOE	2

After running the pipeline with this IK user depended content for the pipeline, the IK motion finally changed: the rotation of the hip changed. As you can see in figure 8 the twist is totally gone.

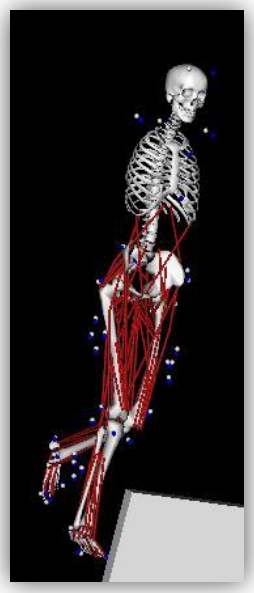

Figure 8: IK motion of the last experiment.