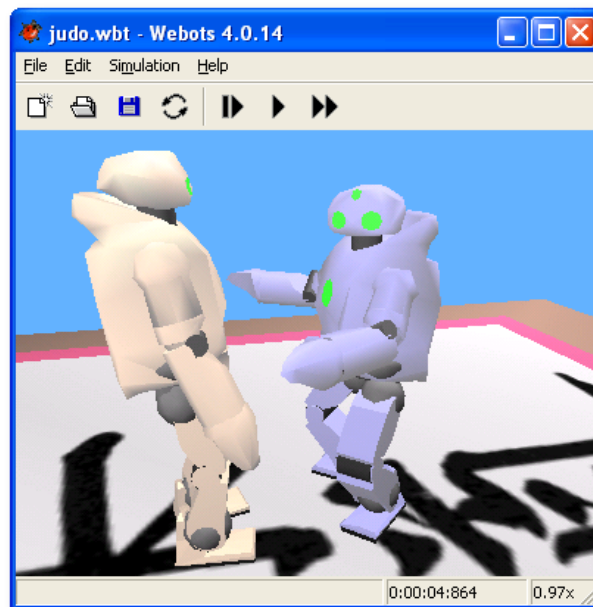


# Simulated Robot Programming Contest



by  
**Jean-Philippe Egger**

Supervisor: Olivier Michel  
Director: Prof. Auke Jan Ijspeert

Semester Project Summer 2004

Biologically Inspired Robotics Group (BIRG)  
School of Computer and Communication Sciences (I&C)  
Swiss Federal Institute of Technology (EPFL)



# Content

1 Introduction .....	3
1.1 Project Goals .....	3
1.2 Organization .....	3
2. Personal Experience as a new Competitor .....	4
2.1 First impression .....	4
2.2 Installation, Registration and first launch .....	5
2.3 Programming support and useful tips.....	5
2.4 Analysis of competitors Experiences and needs .....	5
3. Creation of the Roboka Web site .....	7
3.1 The web site's minimal functions .....	7
3.2 Graphical design.....	7
3.3 Reorganization of the Web site Structure .....	9
3.4 Implementation of a Chat service.....	10
4. Analysis / Design of helpful tools .....	13
4.1 Previous contributions.....	13
4.1.1 File format for interchanging movements .....	13
4.1.2 Graphical Interface for creating movements .....	14
4.1.3 Evaluation.....	14
4.2 Motion generation Tool: A useful feature.....	14
4.2.1 Key Concepts .....	15
4.2.2 Design of the Motion.....	16
4.2.3 Saving and loading Files .....	19
4.2.4 A practical interface for creating Motions .....	23
4.3 Possible improvements.....	27
5. Conclusion.....	28

# 1 Introduction

Since 1998 Cyberbotics Ltd. [1], a leading company in 3D mobile robot simulation, organizes robot simulation contests on its web site. Since 2002 Cyberbotics Ltd. [1] works in close collaboration with the Biologically Inspired Robotics Group (BIRG) [2] of the Ecole Polytechnique Fédéral de Lausanne (EPFL) [3]. In 2003 the International Robot Judo Contest was created, which successfully attracted hundreds of passionate programmers of humanoid robots from all around the world. The goal of the competition was to share the passion and knowledge about robot controllers in a competitive and amusing environment. In this way, challenging scientific work was turned into a much more enjoyable and comparable competition. The environment was produced by Cyberbotics' [1] commercial physics-based robot simulation software. The controllers were programmed in Java, uploaded to the personal account, and every weekday at 14h00 Central European Time the competition was started with the actual controllers. The competition finished the 5<sup>th</sup> May 2004 with the happy winner receiving a brand new Hemisson robot combined with the Webots PRO package.

## 1.1 Project Goals

The success of the 2003 / 2004 competition edition and the continuous ambition to propagate their enthusiasm and knowledge about robotics gave Cyberbotics [1] and BIRG [2] the decisive motivation to go on. Together they decided to reengage their efforts and resources in a new and ameliorated humanoid robot wrestling competition. Therefore, a complete analysis of the overall user-friendliness and documentation of the actual competition was needed. The primary aims of this project were:

- Improve the user-friendliness of the web site and the organization of the competition

- Transfer the competition from Cyberbotics' commercial web site to one of its own

- Encourage new competitors to enter the competition by giving them useful tools

## 1.2 Organization

This project was separated into three main phases as following:

- Analysis of competitors' experiences from their inscription, through their first lines of code and up to their daily development.

- Creation of a web site specially dedicated to the Judoka Competition

- Analysis / Design of helpful tools for facilitating and encouraging new programmers to enter the competition.

## 2. Personal Experience as a new Competitor

Without any further explanations I was asked to surf the competition's web site, to relate my personal experiences with it and to give my criticisms.

### 2.1 First impression

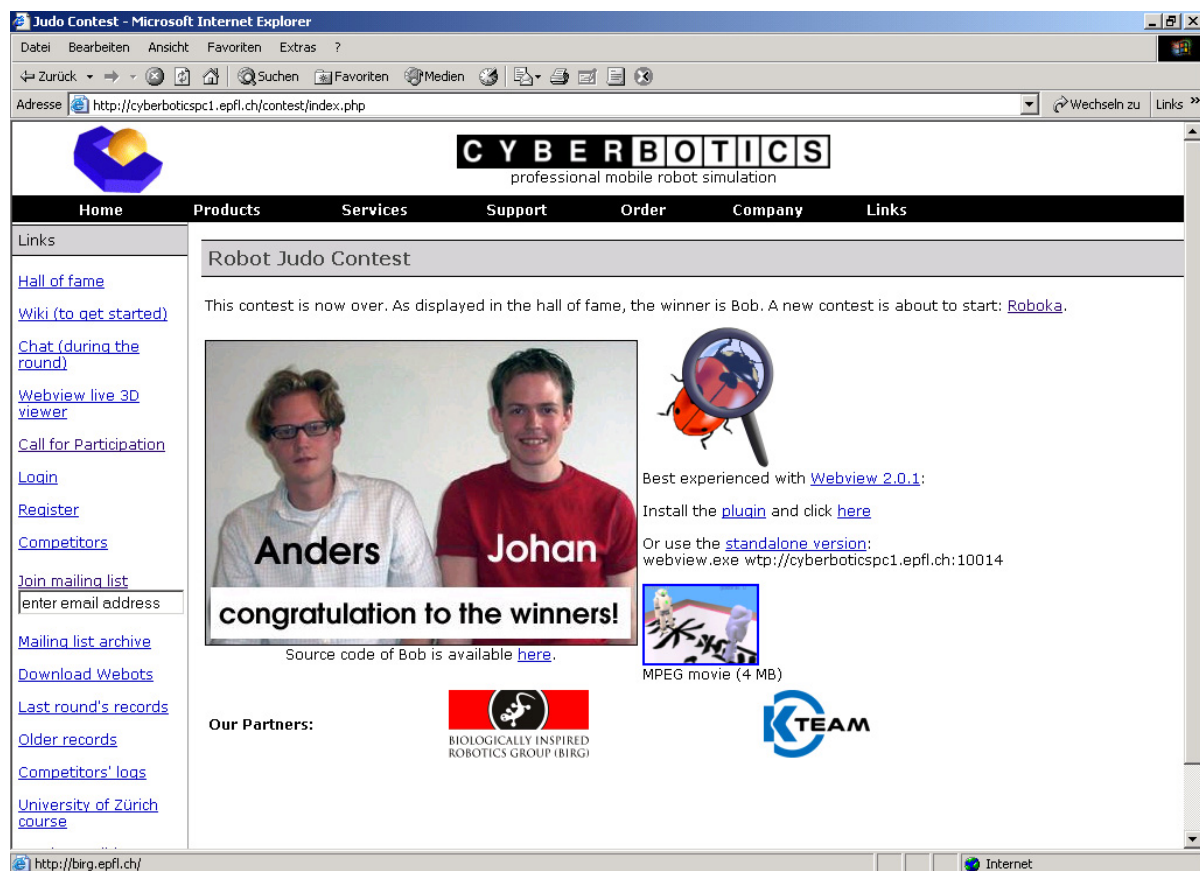


Figure 1: A screenshot of the old competition's web site

The old Judoka contest was still running when I started this project. The home page of the competition was a sub section of Cyberbotics' [1] commercial web site. The competition's nature and environment was instantly recognisable thanks to the image showing the running competition. A new visitor could immediately recognise what the competition was about. The many links on the left of the site were a bit confusing but revealed much useful information. The overall first experience was attractive and centred around the image of the running competition.

## **2.2 Installation, Registration and first launch**

The lack of structure regrouping similar links to the left of the screen made it difficult for a new visitor to orient himself. Especially the unavailability of a link named “getting started” or “introduction” was surprising to me. After I had viewed a few pages the concept of the site was clear. The most valuable section for a newcomer was the Wiki section which had no indication of its content. Once found, the page was of great help. Especially the very well written “How do I get started” section was extremely useful. Following the section’s instructions point by point was very easy and in a few minutes my first controller was running in the Webots environment. The competition registration part was very straightforward and didn’t create any difficulties. I just missed a small note on the “my files” page, explaining exactly which files must be present for my robot to take part in the competition and appear in the hall of fame.

## **2.3 Programming support and useful tips**

The robot controller shipped by default with the webots software is very well suited for a rapid overview of a working controller. Thanks to this example, the initialisation of the different robot parts (Camera, servos, sensors etc.) and the sending of values to the servos are learned in a few minutes. Thus the programmer can very quickly concentrate on designing an efficient and autonomous robot without spending hours on reading how to program a controller for the webots environment.

The design of an efficient controller for a biped robot is a very difficult task. Apart from the camera and distance sensor analysis, programmers are very quickly confronted with moving the robot around. Even the most basic walking movement can very quickly become an immense effort. The old judoka site had no official “Frequently Ask Questions” or “tips and tricks” section which would guide a new competitor in his analysis. Many competitors posted a rich selection of programming issues and tools on the Wiki section. This section is very useful, but also very poorly structured. This is mainly due to the nature of Wiki pages (Anybody can write or modify an article on the Wiki section). The bad organization and the redundancy made it difficult to localise the needed information efficiently. Thus a programmer had to literally fight his way through all the articles to get an overview of the section.

## **2.4 Analysis of competitors Experiences and needs**

In review, I can say that all the necessary information for competing in the robot competition were accessible through the judoka web site. There are various contributions from different sources of various qualities on the Wiki pages. Unfortunately the organisation of the huge amount of information has not been well planned. This will definitively need improvement.

Once introduced to the world of humanoid robot simulation through Webots, the new competitor is quickly lost in the huge number of possibilities. The lack of tools is obvious.

To be effective in the competition a programmer has to master three completely different aspects:

### **Creation of Motions**

To win, the robot has to move. Even basic movements are not trivial to implement. Having to recompile your controller and restart Webots at each modification is very time consuming. One needs hundreds of runs for fine-tuning a single movement.

### **Sensing the environment**

Decoding the information from the camera and the distance sensor are essential for detecting the opponent and the borders of the tatami. Every competitor needs to implement an image-processing algorithm to achieve this goal. Before running any kind of analysis the format of the data coming from the camera must be understood and a pixel-by-pixel navigation must be implemented. For the analysis phase, the competitors must painstakingly find a balance between the speed of the algorithm and its accuracy.

### **Sensing the robots inclination / position**

The GPS Module of the robot is used to detect his inclination. This module can be used in situations where the robot's actual inclination is necessary for making strategic decisions. The detection of a fall or the corrections of a movement are examples of situations demanding the use of this module.

Each aspect is absolutely necessary for successfully evolving in the competition. Mastering only one alone will not permit a competitor to be effective. Only the merging of these three elements, combined with powerful tactical decision-making, will induce intelligent and autonomous robots.

Unfortunately each aspect has its complexities. A new programmer will therefore spend weeks programming, slowly increasing the features of his robot, without yet creating a robot capable of competing in the competition. This initial phase of building all the basic modules for the robot controller can be very frustrating because all the efforts don't result in visible progress. Thus the fun part of the competition, watching the robots competing against another, can be postponed by many weeks.

I can imagine many helpful tools for aiding developers to overcome this initial difficulty. Being able to launch a robot into the competition after only a few days of coding should be very motivating for newcomers. The tools should focus on helping the user to implement movements quickly and on sensing the basic environment. With such tools in hand, any motivated programmer could very quickly evolve into a serious competitor.

## 3. Creation of the Roboka Web site

For the 2004 / 2005 edition of the robot competition, the decision was taken to separate the competition site from Cyberbotics' [1] commercial web site. Thus, I was charged with redesigning and implementing the new web site. We decided to name the new competition the "Roboka Cyber Robot Competition". Therefore the name roboka.org was officially registered.

### 3.1 The web site's minimal functions

The competition's web site is connected to the webots simulation software by its file structure and through various scripts modifying that structure. When running the competition, webots searches specific directories for the competitor's controllers and their respective descriptions. In this scenario, the web site has to assure at least the following functions:

On registration of a competitor to the competition, the scripts on the web site must create the corresponding directory for the competitor's personal files.

A log-in mechanism for recognition of registered competitor must be available

Once identified, a competitor must be able to upload new files and to remove existing ones to his personal folder.

Additionally the web site should also be able to represent the actual ranking of the competition.

### 3.2 Graphical design

The first step in designing the site was to draw a simple draft of the layout. The upper menu structure of Cyberbotics'[1] web site seemed a very pleasant and practical graphical element. Thus, I decided to include it in the new site. The first draft was presented and the general concept accepted.

Next we needed a logo for our website. The first logo was a drawing based on a picture of the original QRIO robot from Sony. The Logo came out quite satisfactorily (see Figure 2).



**Figure 2: The first logo for the web site**

Unfortunately, Sony is very proud of its flagship QRIO and is very sensitive about the use of its image. Already having experienced this issue, Olivier Michel [4] encouraged me to modify the logo to hide the resemblance. I preferred starting from scratch again, convinced that modifying small elements of the present logo wouldn't mask the similarity with the Orion robot. Additionally, it would probably destroy the friendly features that make the robot so attractive.

The second approach was drawn from scratch and the result found wide acceptance:



**Figure 3: The final logo for the web site**



With the Logo drawn and the navigation menu chosen, the graphical structuring of the pages was still pending. My biggest concern was to keep a clean, simple and functional design. I decided to use the same colour for the menus and the paragraph titles. For better separation of different paragraphs on the same page I introduced a screen wide horizontal banner. The final result completely addresses my initial concerns and gives the web site a friendly look while keeping the page structure clean and sober.

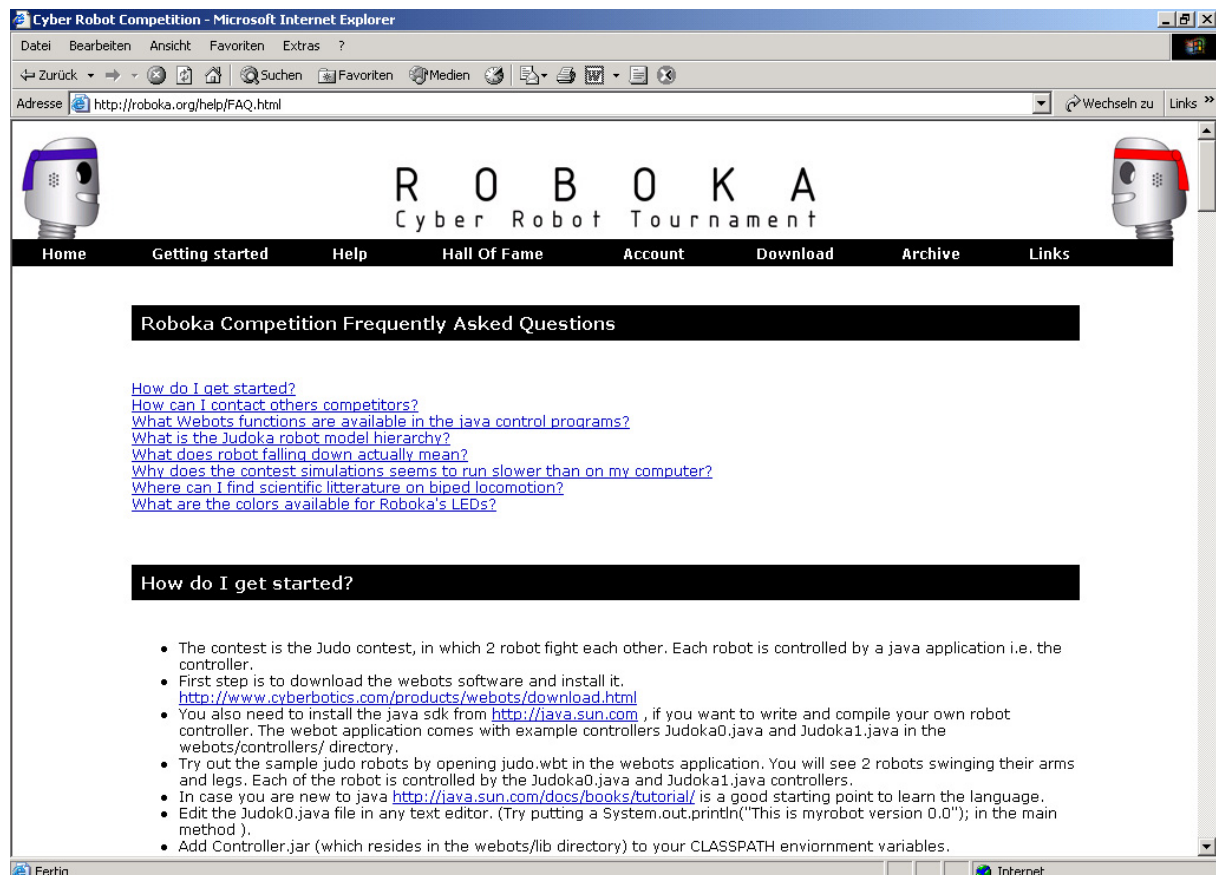
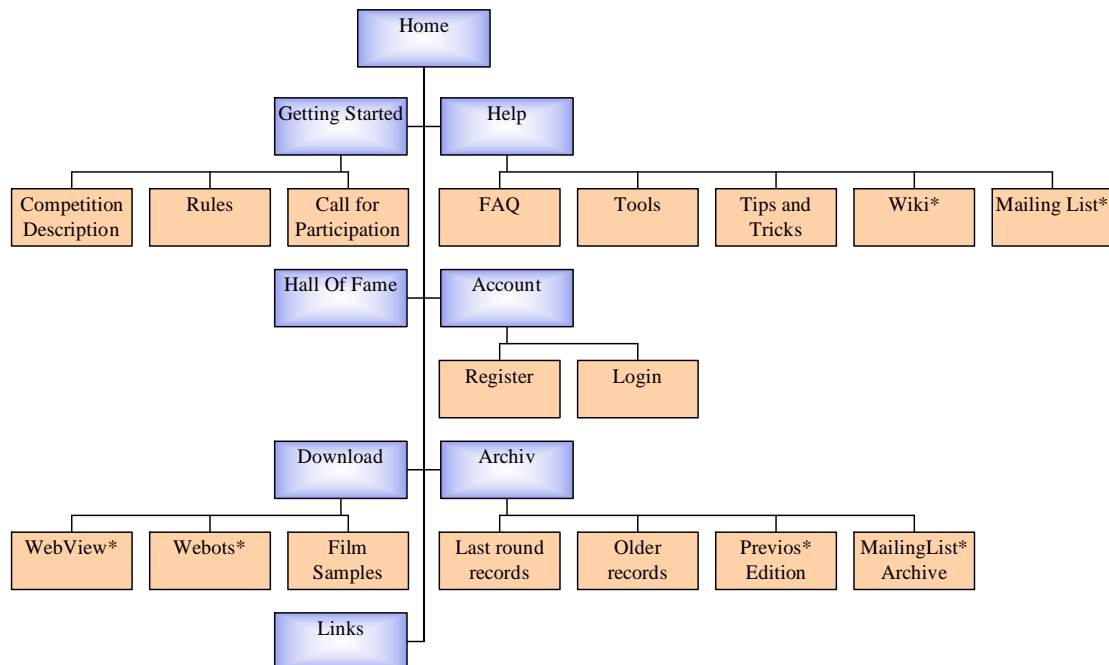


Figure 4: A screen shot of the new web site

### 3.3 Reorganization of the Web site Structure

From my personal experience as a new competitor (see chapter 2), I noticed a lack of “official” documentation. As the new site should address this need, I decided to sort and filter the valuable information from the old Wiki pages and restructure this information for easy accessibility. My goal was to improve the experience of new competitors by increasing their productivity through well-organized documentation.

First of all I needed to create categories to regroup web pages. The global structure of the Web Site is the following.




**Figure 5: The Structure of the Roboka Web site**

The introduction of a structure, regrouping similar pages, considerably increased the overview for the visitor. The external Links (Marked with a star on Figure 5) could simply be copied from the previous edition. All other pages had to be re-formatted to match the new design.

Two major new categories were introduced: “Getting Started” and “Help”. These two sections regroup the most valuable information of the old Wiki Pages. I personally reviewed each contribution, tested them for correctness, selected my favorites, rewrote some parts and finally regrouped them into the web site structure. I am convinced that the new structure and condensed documentation will greatly facilitate the entry of newcomers into the competition. Hopefully this will also encourage them to become serious competitors, keeping this exciting competition a constantly evolving challenge.

### 3.4 Implementation of a Chat service

Communication between the competitors is an important feature. Programming can sometimes be a lonely and time-consuming activity. Having personal contact with others sharing the same passion can greatly increase a programmer’s motivation and satisfaction. Therefore we searched for an easy way to encourage web sit visitors to interact. A chat service, parallel to the competition, instantly seemed a perfect means for achieving our goals.

The complete web site was functioning without a database, so we didn't want to add a database server just for that. A simpler solution had to be found. Oliver Michel [4] quickly came up with a very promising solution named Slack Chat by [MCode@](#). The complete php code was only 35 Lines long.

The chat is mainly php code and is composed of three parts:

1. The file representing the chat text:

Every person participating in the chat writes directly into this file when hitting the submit button. The file is then presented to all the site visitors in the chat window. The window refresh is achieved by adding a simple JavaScript header to the file.

2. The html code creating the "Enter Chat" Button

A standard html button, which opens a new window for the chat

3. The php script:

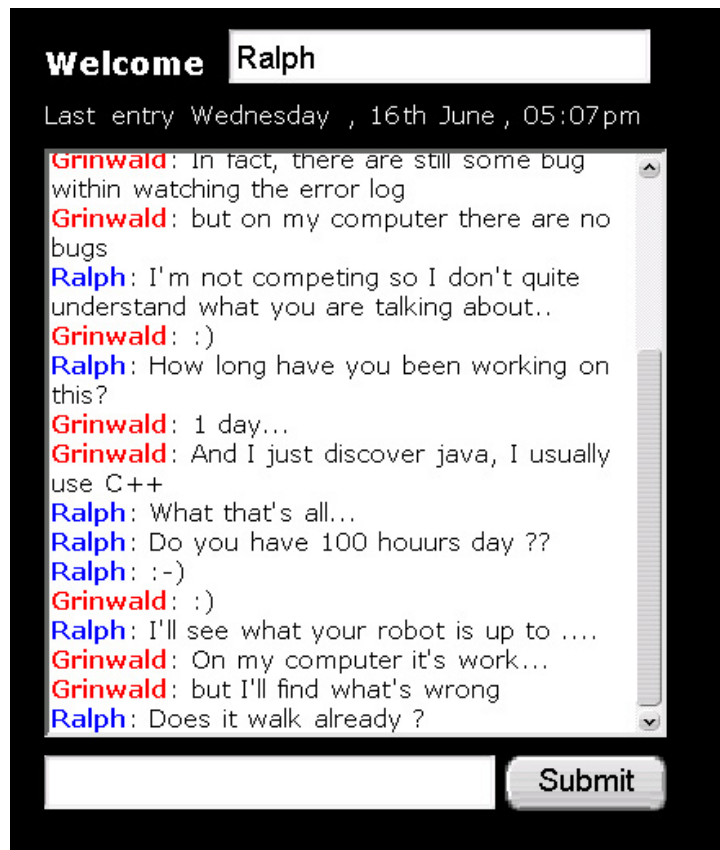
This script creates a submit button tied to a html form which sends the users text when the submit button is hit.

The script displays the chat text in a frame.

Finally, the script also responds to a submit request by opening the file representing the chat text and adding the text received into it.

The structure is so simple it didn't take much effort to incorporate it into the web site. Modifications had to be made because I wanted to completely integrate the chat into the web site, not just open another window. To avoid refreshing the complete page every five seconds I integrated the php code into a separate html iframe. The chat was quickly up and running. Unfortunately there was no scrolling function implemented. This meant that after a certain number of chat messages (default was 10) the complete discussion was erased and a new one started. This was very impractical and frustrating behaviour. To implement a scrolling of the text, I had to modify the response to a "submit". Instead of just appending a line to the text file, the complete chat discussion is now loaded from the file, the first line removed, the new text appended and the old file overwritten. After some effort this was successfully implemented and can be experienced on the site.

We thought it would be nice to differentiate logged user from conventional site visitors in the chat window. Upon login a cookie is written on the competitors computer. The recognition was already implemented for activating the users access to his personal files. I reused this recognition for modifying the chat panel in real time. A logged user cannot enter his name for chatting. He can only chat with his login name. When entering a message a logged user will appear red in the chat box while site visitors have a blue colour. In this way competitors are identifiable by every one and nobody with a red colour can pretend to be another competitor.



**Figure 6: The chat window: Logged competitors appear in red, others in blue**

The last feature I added to the chat was the time stamp of the last chat message. This is done by reading the last modified date of the file while constructing the chat window. This helps people to know if other people might still be online.

The last step was to graphically incorporate the chat window to match the web site's design. I decided to place the chat window right next to the competition view. This increases intractability, permitting all the visitors to watch a match and follow the real time discussion simultaneously. The chat window blends into the web site surprisingly well. Using the same colour once again, the chat doesn't attract too much attention and keeps the overall view clean.

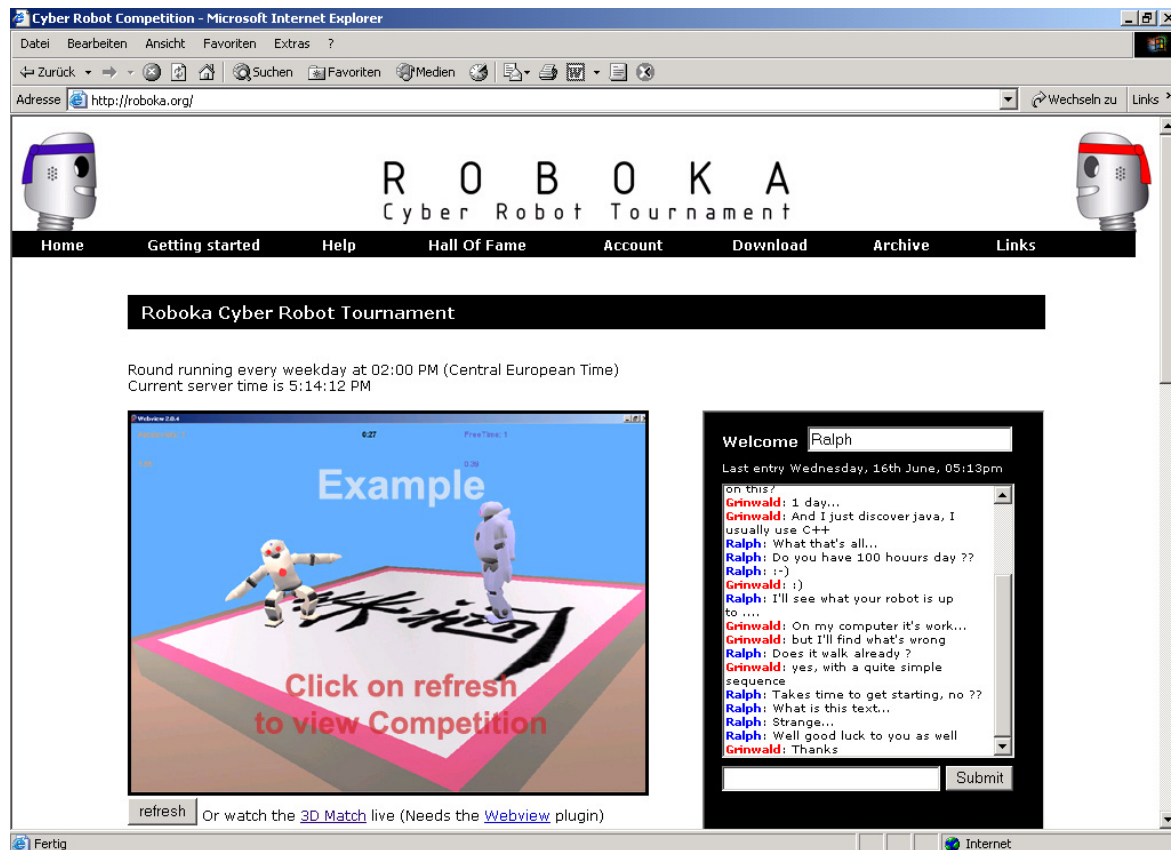


Figure 7: A Screen Shot of the new Home Page with integrated Chat

## 4. Analysis / Design of helpful tools

As described in chapter 2, my personal experience as a newcomer revealed a few deficiencies. After improvement of the structure of the website, the programmers needs should be better served. For this reason many competitors posted very interesting concepts and tools on the old Wiki pages.

### 4.1 Previous contributions

Two articles posted by Yves-Marie Le Dru (Albathor) and Xavier Bitot (Kubiak) respectively instantly attracted my attention. One of them proposed a human readable file structure for defining movements while the other had programmed an interface for creating movements.

#### 4.1.1 File format for interchanging movements

Yves-Marie's file format seemed a very interesting idea to me. The motion's definitions are stored in a human readable text file, the format resembling to a XML structure. The motion is decomposed into key frames, which at a certain time define the positions of all the servos.

Yves-Marie included a parser to his contribution, which parses the file and loads the movements into memory. The motion is then executed by interpolating the position of each servo key frame. With this concept, the motion definition is completely separated from the controller executing the movement. This is a very interesting concept which seemed very well adapted for a community, leaving programmers the possibility of using their own Interpolation methods. Further, this separation can be used for sharing motions, thus enabling new competitors to benefit from previous contributions and to develop better skills much faster.

#### **4.1.2 Graphical Interface for creating movements**

Xavier's Interface for creating movements was the only really usable tool that was posted on the Wiki Pages. The interface was composed of a drawing board, a result board and the possibility of defining trajectories for different servos. Thus a programmer could select a servo, draw its trajectory on the drawing board and instantly see the robot's behaviour by hitting the GO button. This tool increases the productivity of movement creation enormously. No need to recompile your controller and re-launch webots at every modification. Simply draw the trajectories, click on the go Button, and observe the robot's motions.

#### **4.1.3 Evaluation**

Both contributions have very interesting concepts. Unfortunately there is no link between them. Xavier used a very large array to store each servo's position at every time step during the movement. Thus, when a movement was saved, the file had a considerable size. Yves-Marie's concept of key frames for defining motions seemed much more powerful. His definition of a key frame is quite rigid in the sense that at one moment in time the programmer has to define all servos' positions. Typically a servo that isn't used is set to zero. I didn't like this concept because of a majority of movements using only a few servos. I could imagine combining a motion of scanning (head) with a walking motion. In Yves-Marie's design, this is not possible since one movement overwrites all the servo values. Also imagine the robot is in a specific position. If you want to keep this position and only move the robot's head, you would have to create a motion where at each key frame the actual positions of the servos are repeated. This concept is very restrictive and forces the programmer to think about all servos even though his motion involves only a few. Thus, these contributions were very inspiring and I could imagine many improvements.

### **4.2 Motion generation Tool: A useful feature**

Inspired by the two contributions of Yves-Marie and Xavier, I decided to engage myself in creating a movement generation tool designed for a community of programmers, each having his own concepts. I aimed at providing an easy and efficient way to create movements and to encourage programmers to share some of their valuable work. I am fully aware that competitors will not want to share their most valuable motions. Only basic contributions like walking or maybe a stand up movement would immensely facilitate a new comer's entry into the competition.

### 4.2.1 Key Concepts

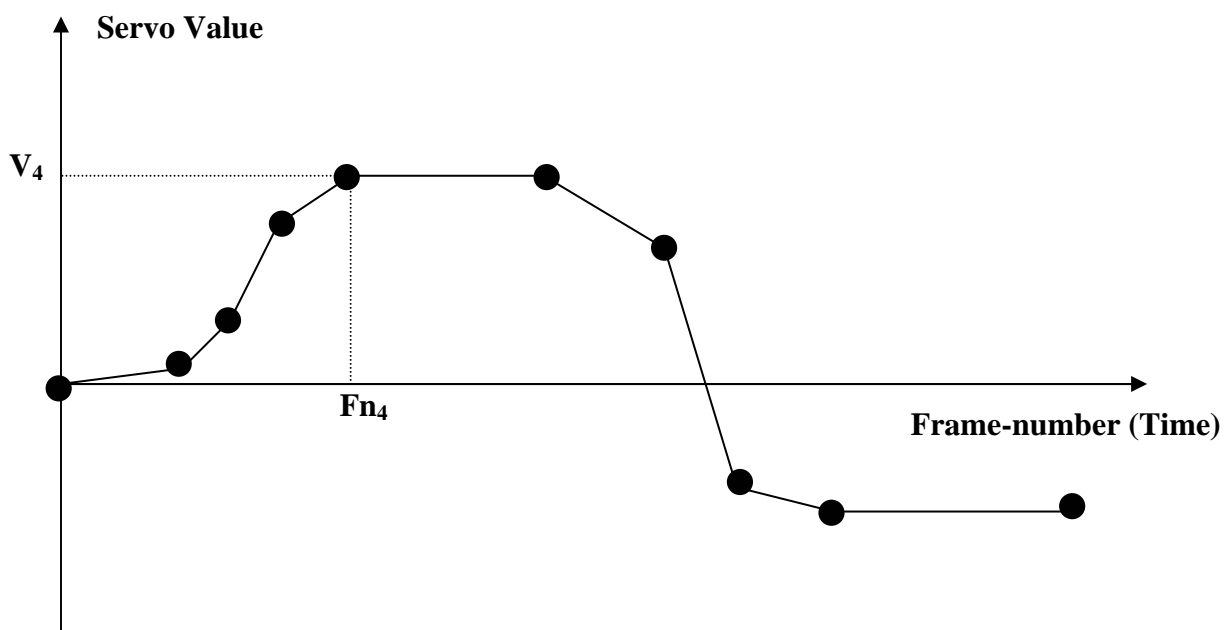
So the analysis started on how to achieve my goals and how to best serve the programming community. The idea I had in mind was to use standard XML format for saving motion definitions into files. The XML standard is a universally accepted technology and its application field is growing with each day. Many free and commercial tools exists for retrieving / manipulating an information structure defined in XML.

Before even thinking of a means for saving and sharing motion definitions, I had to design a portable and flexible concept for modelling motions. This concept is the heart of my application and is the foundation on which any further construction will be based.

By definition, a motion in robotics is created by sending position values to the robots servos. The set of all the positions for a certain servo over a time window is called a trajectory. The combination of trajectories from different servos defines a movement. In Webots, time is simulated in discrete steps of 32ms. This means that every 32ms the servos can take a new position. From now on, these discrete steps will be called frames.

No programmer wants to define the position for each servo at each frame. A small calculation shows that for a movement which takes three seconds to execute, a programmer would have to set about 100 values for each servo. Thus, a more general way of defining trajectories must be available.

The Key Frame concept is much more efficient for defining a trajectory. A key frame defines what position a certain servo must have at a specific time. Combining key frames and interpolating the positions in-between them, will completely define a trajectory (See Figure 8).



**Figure 8: Combination of key frame (points) and its interpolation forming a trajectory**

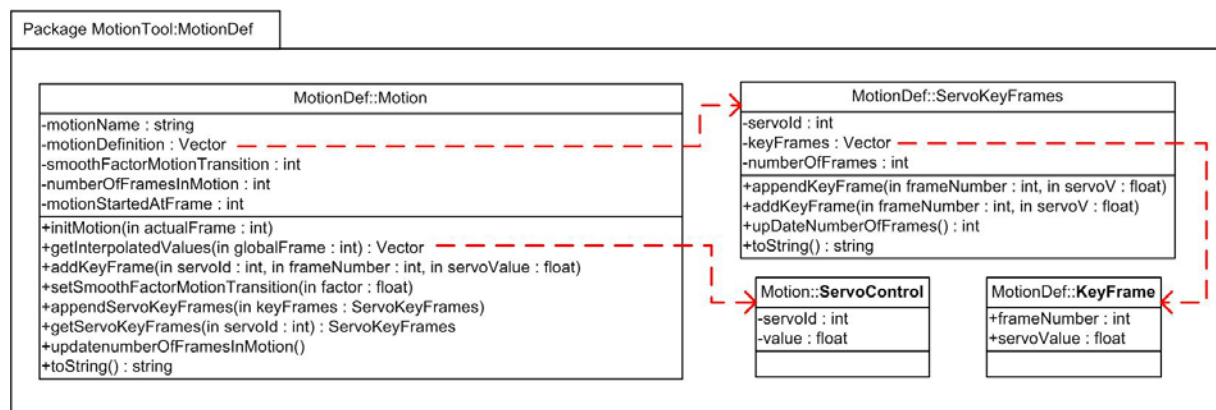
Defining a motion will thus be composed of creating trajectories for each servo involved in the motion. A trajectory will be further decomposed into an unlimited number of key frames. Each key frame is attributed to a single servo and defines the time (in frames) at which to reach a given position. Once a motion is defined, an Interpolator (linear or not) must be implemented to calculate the values for a servo at any given frame.

#### 4.2.2 Design of the Motion

To keep the motion creation as flexible as possible, I didn't want to force a programmer to define key frames at the same time interval for each servo. A servo should be completely independent in its trajectory from others.

A second important point was the ability to define only the servos involved in the motion, not having to bother with the others. This is implemented by sending values only to the servos that have at least one Key Frame defined. Thus in a motion were only the head is involved, the other servo will keep the positions they had at the beginning of the motion. This enables a robot to keep a certain position while moving certain other parts of his body. The position can be arbitrary and is completely independent from the motion, which will be executed correctly.

The Motion was decomposed as following:



**Figure 9: The structure of the MotionTool.MotionDef package**

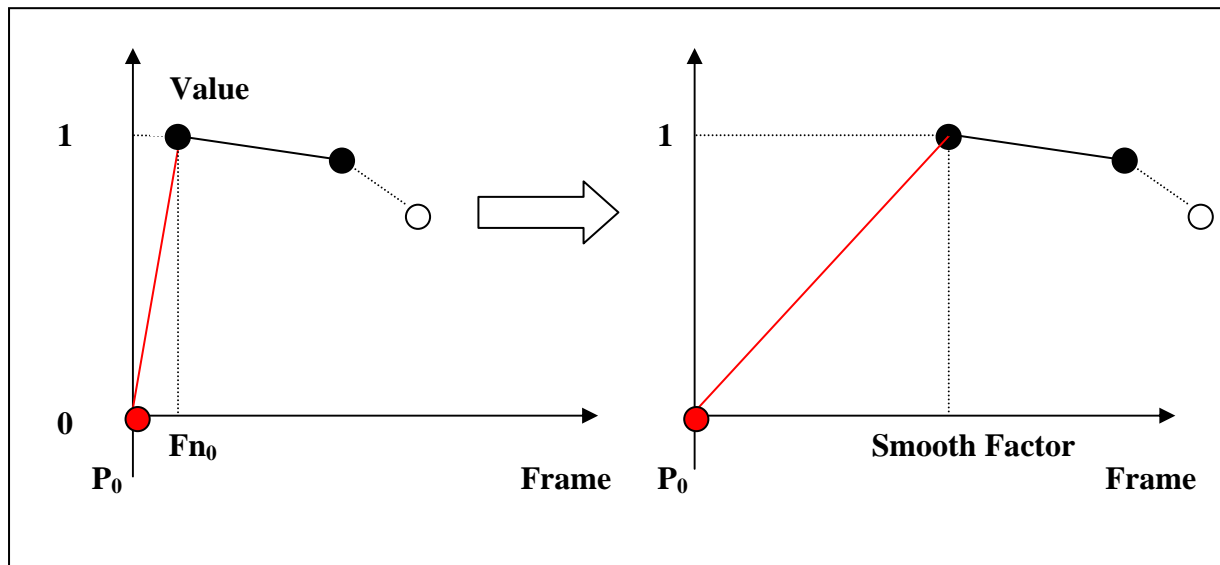
A motion is composed of a vector of servo-key-frames. The **ServoKeyFrames** class gathers the trajectory definition for a single servo. Therefore it has a servo-id attribute and a vector of key-frames. The **KeyFrame** class simply regroups servo Value and frame position. The attribute **frameNumber** of the **KeyFrame** class represents the value  $F_{n_4}$  in figure 8 while the attribute **servoValue** represents  $V_4$ .

The Motion class is used as follows: The default constructor is used to create the object. Once created the function **addKeyFrame (servoId, frameNumber, servoValue)** is called as many times as necessary. The smooth factor can be set to 20 by using the function **setSmoothFactorMotionTransition(20)**.



The smooth factor is used to smoothen the transfer from the actual position to the beginning of a motion. It defines how many frames (discrete time steps) are necessary for a change of servo Value of one. If set to 0, no smoothening will take place.

Example **SmoothFactorMotionTransition** = 20: If the actual position of the left hip is 0.3 and the first key frame of the left hip has a servo value of -0.7, the Interpolator will add  $(0.3+0.7)*20= 20$  frames to pass from the actual position to the beginning of the motion. This means the start of the motion will be shifted by 20 frames (see Figure 10)



**Figure 10: Smoothening the transition to a new movement**

Once the motion is defined it has to be initiated with the actual global frame number using the **initMotion(globalFrameNumber)** function. This sets the start of the motion to the actual frame number and calculates the number of smoothening frames necessary.

The interpolated values of each servo can now be obtained by calling every 32ms the **getInterpolatedValue(globalFrameNumber)** with the **globalFrameNumber** incremented at each call. This function returns a vector of servo values, which must be sent to the servos. See the code in Figure 11 for an example of how to use the motion class.

```
//Create a motion object
Motion actualMotion = new Motion("jump");
//Add Key frames
actualMotion.addKeyFrame(0,12,0.3f);
actualMotion.addKeyFrame(0,25,0.8f);
actualMotion.addKeyFrame(0,56,-0.7f);
.
.
.
actualMotion.addKeyFrame(13,0,0f);
actualMotion.addKeyFrame(13,10,0.3f);
actualMotion.addKeyFrame(13,90,-0.3f);

//Init the global Frame number(is equivalent to time)
int globalFrame = 0;

//Set the smooth Factor
actualMotion.setSmoothFactorMotionTransition(20);

//Init the movement with the actual global Frame Number
actualMotion.initMotion(globalFrame);

for(;;){

    //Get the interpolated values
    Vector servoPositions = actualMotion.getInterpolatedValues(globalFrame);

    try {
        //Each element in the servoPositions vector is a servoid / Position
        //couple to be sent to the servos
        for (int i = 0; i < servoPositions.size(); i++) {
            Motion.ServoControl control = (Motion.ServoControl) servoPositions.elementAt(i);
            servo_set_position(joint[control.servoid], control.value);
        }

    } catch (Exception e) {
        System.out.println(e.toString());
    }

    globalFrame++;
    robot_step(32);
}
```

**Figure 11: Example of how to use the Motion Class**

The motion class doesn't directly send the values to the robot controller. This was designed intentionally to allow for the possibility of correcting the interpolation of the motion at run time. I can easily imagine adding a correction module to the controller, which senses the robots equilibrium state (for example by projecting the robots gravity centre onto the floor) and modifies the static motion at run time in order to achieve balance.

### 4.2.3 Saving and loading Files

Now that the Motion concept is defined the file format for saving motions can be elaborated. As explained in chapter 4.2.1 I aimed at using XML standards for the files. This file format was chosen because XML is a universally accepted technology and because its application field is growing every day. XML is easy to read and purely text based. Thanks to the many XML tools already existing today, manipulating and parsing a XML file becomes very easy.

#### 4.2.3.1 XML Motion Data Interchange Format – XMDI

The goal of this standard is to provide a method for storing complex Motions, representing several trajectories of servos over time, and to do so in a generic, interoperable and extensible manner.

The Structure of a Motion definition conforming to the XMDI is as following:

```
<?xml version="1.0" encoding="UTF-8"?>
<MotionTool xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=".MotionTool.xml">

  <MotionDef>

    <Motion name="Jump" defaultLoop="1">
      <servo name="back_1">
        <positions>1 .3 -.4</positions>
        <frames> 0 20 60</frames>
      </servo>
      <servo name="back_2">
        <positions>0 -.7 .2</positions>
        <frames> 0 10 40</frames>
      </servo>
    </Motion>

    <Motion name="Scan" defaultLoop="1">
      <servo name="left_knee">
        <positions>1 .7 .4</positions>
        <frames> 0 20 60</frames>
      </servo>
      <servo name="right_knee">
        <positions>0 -.7 .2</positions>
        <frames> 0 10 40</frames>
      </servo>
    </Motion>

  </MotionDef>

</MotionTool>
```

**Figure 12: XML Motion Data Interchange Format – XMDI: Example of a motion definition file with two simple motions (Jump and Scan).**

The elements of XMDI are classified into two distinct groups. Containers and Objects.

#### 4.2.3.2 XMDI Objects

XMDI objects are entities, which contain the position information. These are:

```
<positions>
<frames>
```

All XMDI objects shall not contain any attribute. Its content is a single text representation of several floating point values.

Examples:

```
<positions>0 -.7 .2</positions>
<frames> 0 10 40</frames>
```

#### 4.2.3.3 XMDI Containers

Containers are groups of XMDI objects. The containers are:

```
<MotionTool>
<MotionDef>
<Motion>
<servo>
```

##### **<MotionTool> Container**

The root node of the file of which there can only be one. It is a container for <MotionDef> nodes.

Example:

```
<MotionTool>
  <MotionDef>
    ...
  </MotionDef>
</MotionTool>
```

**<MotionDef> Container**

It is a container for <Motion> nodes.

Example:

```
<MotionDef>
  <Motion>
    ...
  </Motion>
  <Motion>
    ...
  </Motion>
</MotionDef>
```

**<Motion> Container**

This element contains only servo elements. It must have the name attribute. Further attributes can be assigned to it. Programmers are encouraged to add as many attributes to their motion as they feel is necessary.

Example:

```
<Motion name="Jump" height="12">
  <servo>
    ...
  </servo>
  <servo>
    ...
  </servo>
</Motion>
```

**<servo> Container**

Unlike the other containers, servo must contain exactly two XMDI objects: The servo element must be composed of a single position object followed by a frames object. The name attribute must be defined. Further the numbers of position values must be equal to the numbers of frame values.

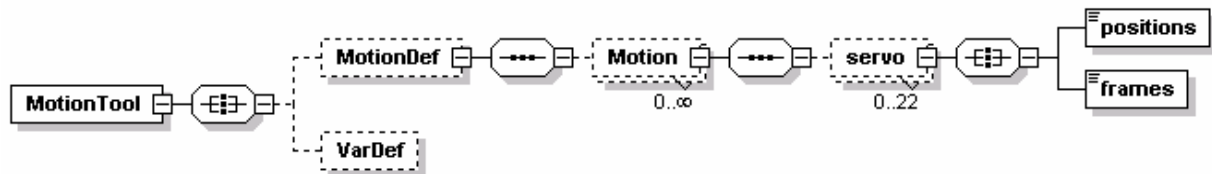
Example:

```
<servo name="back_2">
  <positions>0 -.7 .2</positions>
  <frames> 0 10 40</frames>
</servo>
```

#### 4.2.3.4 XMDI Style Sheet

Valid XMDI files **MUST** conform to the XML Schema represented in figure 12.

The style sheet is included with the motion tool. Any document conforming to this style sheet will be correctly loaded.

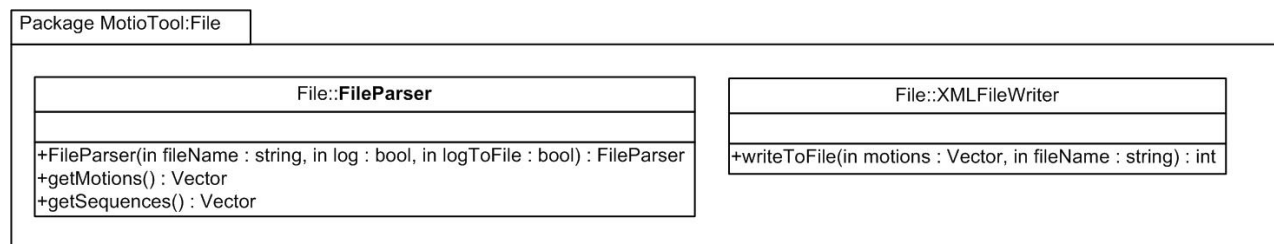


**Figure 13: A visual description of the XML Schema defining XMDI**

#### 4.2.3.5 XMDI Parser / Writer

For loading and saving a Motion to a file I had to implement a file parser and a file writer.

The **FileParser** class and the **XMLFileWriter** class are part of the **MotionTool.File** Package.



**Figure 14: The MotionTool.File Package**

Parsing a file and retrieving the created motions is very easy using the **FileParser** class

```

FileParser parser = new FileParser (filename,true,true);

If (parser.hasParsedErrors) System.out.println("There was an error Parsing the file. Please view the errors
in the errorLog.txt / debugLog.txt files");

else Vector loadedMotions = parser.getMotions();
  
```

**Figure 15: Example of using the FileParser class**

Writing a Vector of Motions into a correctly formatted XMDI-file is done by using the **XMLFileWriter** class:

```
XMLFileWriter writer = new XMLFileWriter();

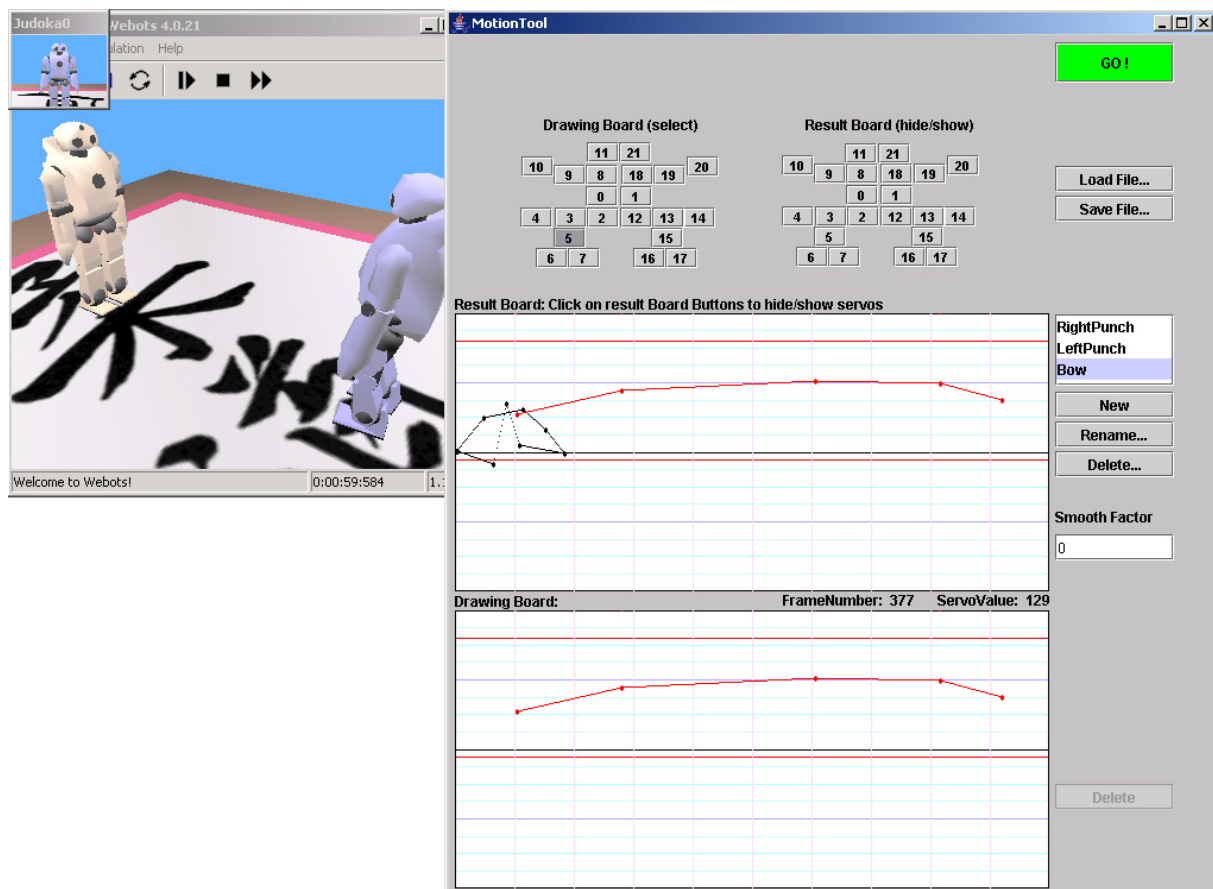
If (writer.writeToFile(motionVector, filename) < 1) System.out.println("Error while writing the file ");

Else System.out.println("Motion successfully written to " + fileName );
```

**Figure 16: Example of using the XMLFileWriter class**

#### 4.2.4 A practical interface for creating Motions

The Key Feature of the Motion Tool application is the possibility of dynamically creating motions by literary drawing them on a board. The motion created this way can be directly visualized by hitting the GO Button. No further compilation is needed and webots doesn't have to be restarted. The graphical part of the interface is strongly inspired from Xavier's interface.



**Figure 17: Screen shot of the MotionTool graphical Interface**

#### **4.2.4.1 Using the interface**

The interface is composed of a drawing board (lower board), a result board (upper board) and various buttons to control the actions. The drawing board is used to draw trajectories of a single servo, while the result board displays all the trajectories superposed. The buttons forming a robot on the top of the interface are connected to the servos. The left group of buttons selects which servo is being edited on the drawing board while the buttons in the right group select which servos are displayed on the result board. In a few simple steps a motion can be defined and tested in the webots environment:

##### **Selecting the actual servo for drawing**

The drawing servo is selected by using the drawing board selection buttons on the top left corner of the interface

##### **Creating a Key Frame**

Clicking on the drawing board creates key frames for the selected servo. Useful information about the key frame is displayed to the right when moving the mouse cursor over it.

##### **Manipulating Key Frames**

Once a key frame is created it can be dragged using the mouse. Clicking directly on a key frame activates the delete Button. Clicking on the delete Button when activated will erase the selected key frame.

##### **Creating a trajectory**

A trajectory is composed of key frames. Just select the servo for which you want to define a trajectory and start creating key frames. A line on the drawing board directly displays the interpolated result.

##### **Viewing the composed trajectories**

Which trajectories are visible in the result board is chosen by activating/deactivating the result board buttons on the upper right.

##### **Running / stopping the execution of a motion**

Hitting the GO button in the upper right corner launches the motion in the webots environment. The actual motion is executed following the trajectories defined in the interface. When running the simulation, the GO button is turned into a STOP button. Clicking on it will reposition the robot at the centre of the Tatami and stop the simulation.

##### **Loading / saving motions to a file**

Use the Load File button to load a motion from a file. Any motion actually defined will be erased and the new motions loaded to the interface. Use the save button to save the motions.



#### 4.2.4.2 The MotionTool.Interface package

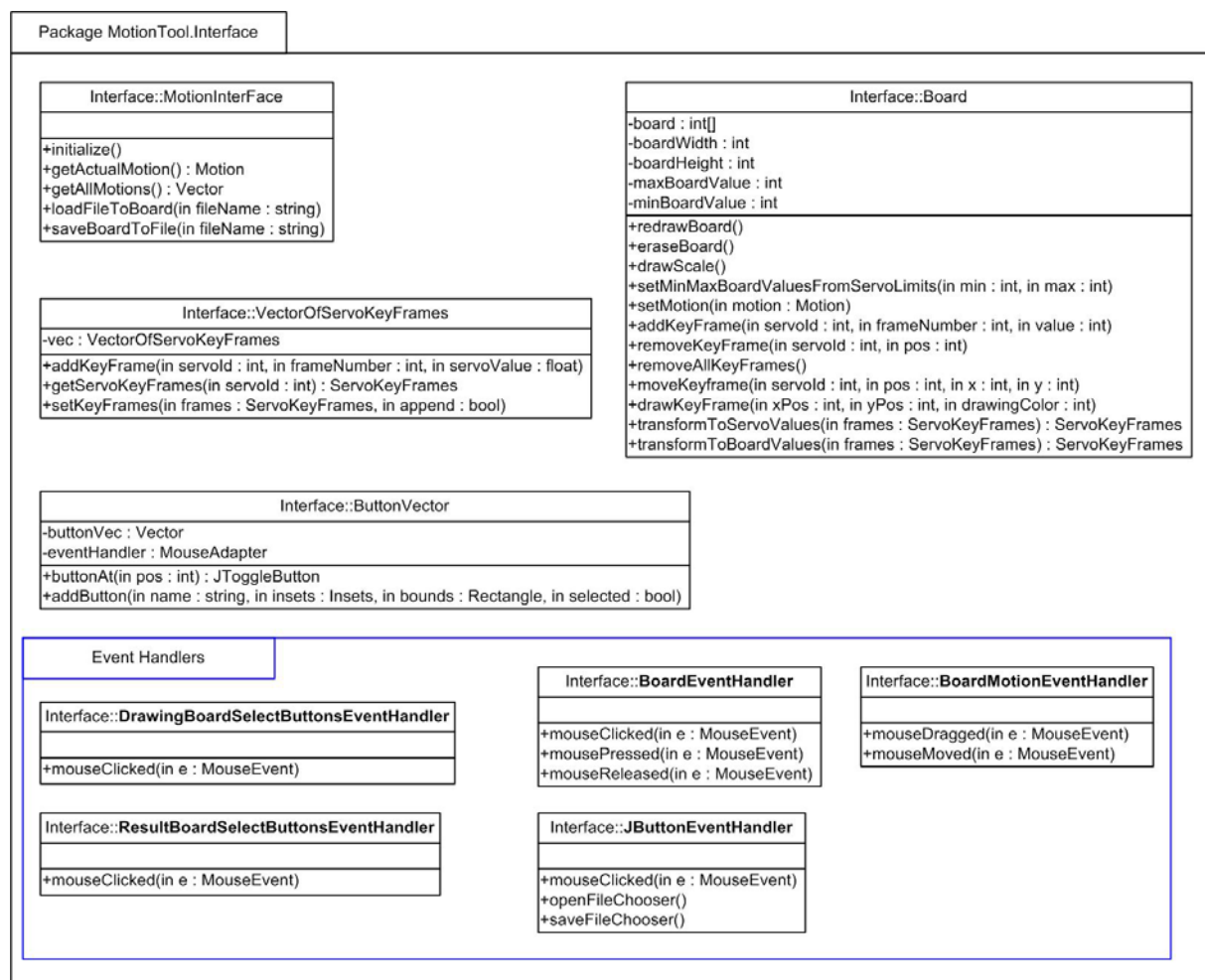
The interface was separated into four main classes.

**MotionInterface.java:** The swing interface with all the graphical elements

**Board.java:** A graphical drawing board constructed around the key frame concept

**ButtonVector.java:** A vector of buttons representing the select / unselect buttons of the drawing / result boards

**VectorOfServoKeyFrames.java:** This class is used to store the key frames of the actual motion in the boards.



**Figure 18: The MotionTool.Interface package**

The event handlers react to the mouse and key events of the interface. The format of the key frames for the drawing / result board is not the same as used by the Motion class to interpolate the trajectories. This is due to the mapping of servo values ( $-3.14$  to  $3.14$ ) to the board coordinates ( $0 - 234$ ). For this purpose, transformation functions in both directions are available in the board class.

#### 4.2.5 Final Design of the Motion generation Tool

The Motion tool application is composed of three main parts: The **MotionDef**-package, the **File**-package and the **Interface**-package. The **MotionDef** and the **File** packages are completely independent from the other packages. Thus, motions can be created and used in the **Jucoka** controller without using the interface or the file parser. This separation gives a programmer the possibility of implementing his own **Motion** classes while still using the Interface to create the final motions. Especially rewriting the interpolation function or the smooth factor behaviour could be of use.

The interface package regroups all the functionalities of the **MotionDef**- and **File**-packages to a single application. The graphical interface is launched by initialising the interface object inside the Judoka controller.

The interaction between the different parts of the **MotionTool** is displayed in figure 17:

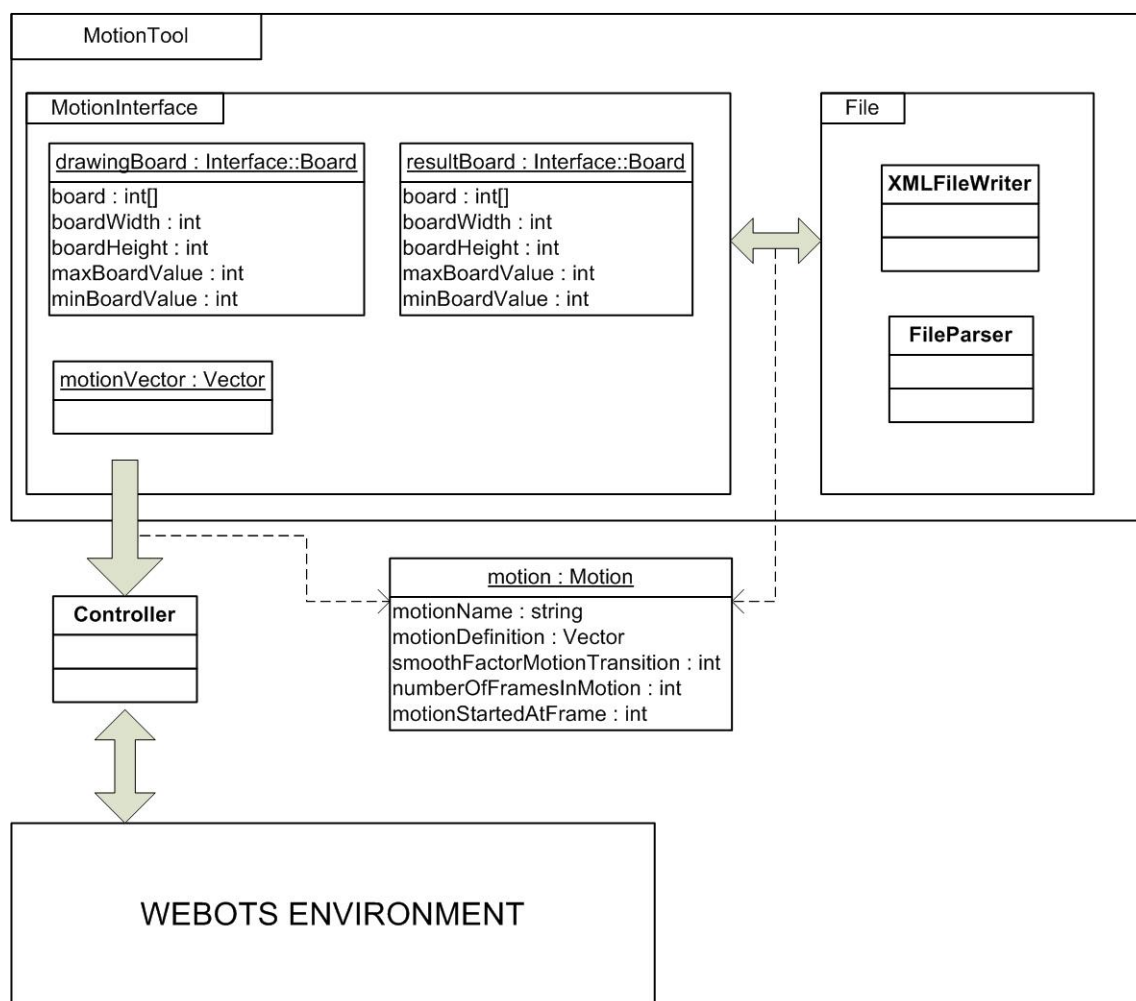


Figure 19: Overview of the Motion tool application

### 4.3 Possible improvements

The Motion tool application was designed to facilitate future expansions. The core of the application is now settled, but many additional features could still be added to the interface. Here some simple improvements that could be made:

#### 4.3.1 A sequence manager module

Adding a sequence manager module for combining motions to form higher-level behaviors could be very useful. It should be pretty simple to add a new **Jpanel** for this module. Being able to add parameters (like number of times to execute, direction (normal/reverse), etc.) to each motion in a sequence could be a very interesting feature for creating complex executions rapidly.

#### 4.3.2 Some simple tools for improving the drawing experience and speed

A smoothen button to smoothen the actual trajectory of the drawing board.

A stretch trajectory button to stretch the selected trajectory.

A mirror button to copy the trajectories of the robot's left side to its right side and vice versa.

#### 4.3.3 A non-linear motion interpolator combined with Bezier curves for drawing

A much used and efficient way of drawing curves is by using Bezier curves. This technique is used by professional image processing applications like Adobe Illustrator. To implement this technique a key frame must be enhanced by two weights and the interpolator redesigned to calculate these trajectories.

#### 4.3.4 Prolong the drawing time:

At the moment, the maximum length of a movement is 500 frames (16 simulation seconds). Enhancing this by implementing a change of the beginning and end time of the board shouldn't be too difficult.

#### 4.3.5 A motion correction module

The motion class doesn't directly send the values to the robot controller. This was designed intentionally to permit the possibility of correcting the interpolation of the motion at run time. I can easily imagine adding a correction module to the controller, which senses the robots equilibrium state (for example by projecting the robots gravity centre onto the floor) and modifies the static motion at run time in order to compensate a balance problem.

I

## 5. Conclusion

The Roboka Cyber Robot Competition is a great contribution to the robotics community. Participating is a lot of fun, but remains a difficult and time-consuming activity. The new web site improves the localisation of valuable information, which should greatly increase the productivity of new competitors. The new chat window incorporated into the home page increases the interactivity between the different competitors and the site visitors, hopefully creating tighter ties between them. I hope the new overall design and structure of the web page will please the web community and help to attract even more visitors.

The Motion Tool created during this semester project is fully functional and is available on the web site. The difficult task of creating motions is greatly facilitated and shortened thanks to this tool. Even though motion creation is only a part of programming an autonomous Roboka, I am confident that helping newcomers to perform at least this task more easily will encourage many programmers to enter the competition. Especially because motions are the most active part of the simulation, I think a rudimentary robot could even compete without sensing the environment at all.

I am very pleased to have achieved all the goals I was set. The web site is running well and the tool achieved the planned functionalities. Unfortunately, given the limited number of hours in a day, I was not able to compete seriously in the competition and advance on this project simultaneously. The tools are now ready to be used and I hope soon to find time to prove their utility.

I wish all future competitors good luck and hope to encounter them soon in an exciting match at the Roboka Cyber Robot Competition.

## Figures

Figure 1: A screenshot of the old competition's web site .....	4
Figure 2: The first logo for the web site .....	8
Figure 3: The final logo for the web site .....	8
Figure 4: A screen shot of the new web site .....	9
Figure 5: The Structure of the Roboka Web site.....	10
Figure 6: The chat window: Logged competitors appear in red, others in blue.....	12
Figure 7: A Screen Shot of the new Home Page with integrated Chat .....	13
Figure 8: Combination of key frame (points) and its interpolation forming a trajectory.....	15
Figure 9: The structure of the MotionTool.MotionDef package.....	16
Figure 10: Smoothing the transition to a new movement.....	17
Figure 11: Example of how to use the Motion Class .....	18
Figure 12: XML Motion Data Interchange Format – XMDI: Example of a motion definition file with two simple motions (Jump and Scan). .....	19
Figure 13: A visual description of the XML Schema defining XMDI .....	22
Figure 14: The MotionTool.File Package .....	22
Figure 15: Example of using the FileParser class .....	22
Figure 16: Example of using the XMLFileWriter class .....	23
Figure 17: Screen shot of the MotionTool graphical Interface .....	23
Figure 18: The MotionTool.Interface package .....	25
Figure 19: Overview of the Motion tool application.....	26

## References

- [1] Cyberbotics Ltd.: A leading company in 3D mobile robot simulation, based in Lausanne: [www.cyberbotics.com](http://www.cyberbotics.com)
- [2] Biologically Inspired Robotics Group (BIRG): [www.birg.epfl.ch](http://www.birg.epfl.ch)
- [3] École Polytechnique Fédérale de Lausanne (EPFL) : [www.epfl.ch](http://www.epfl.ch)
- [4] Olivier Michel: Founder of Cyberbotics Ltd. [1] and responsible for the technology, marketing and sales of Cyberbotics.