An Interactive Tool for Designing Complex Robot Motion Patterns

Georgios Pierris and Michail G. Lagoudakis

Abstract— **Low-cost robots with a large number of degrees of freedom are becoming increasingly popular, nevertheless their programming is still a domain for experts. This paper introduces the *Kouretes Motion Editor* (KME), a freely-available interactive software tool for designing complex motion patterns on robots with many degrees of freedom using intuitive means. KME allows for a TCP/IP connection to a real or simulated robot, over which various robot poses can be communicated to or from the robot and manipulated locally using the KME graphical user interface. This portability and flexibility enables the user to work under different modes, with different robots, using different host machines. KME is originally designed for the Aldebaran Nao humanoid robot which features a total of 21 degrees of freedom, but can be easily customized for other robots. KME has been employed successfully by *Kouretes*, the RoboCup team of the Technical University of Crete, for designing various special actions at the RoboCup 2008 competition (Standard Platform League).**

I. INTRODUCTION

Low-cost robots with a large number of degrees of freedom are becoming increasingly popular, spanning all ranges from entertainment to education and research. Such examples include the Robotis Bioloid, the Hitec Robonova, the Sony Aibo, the RoboSoft Robudog, the Boston Dynamics Little Dog, the ZMP Pino, the Robotis Uria, the VStone Vision, the Fujitsu Hoap, and recently the Aldebaran Nao robots. Nevertheless, programming such robots beyond trivial combinations of precompiled actions is still a domain for experts. This is dictated by the fact that most non-trivial motion patterns require simultaneous move of several robot joints, while satisfying at the same time dynamic, kinematic, safety, and stability constraints.

This paper introduces the *Kouretes Motion Editor* (KME), an interactive software tool for designing complex motion patterns on robots with many degrees of freedom using intuitive means. The main idea behind KME is the ability to generate, capture, store, manipulate, edit, replay, and export sequences of complete robot poses, which resemble the desired complex motion pattern. KME allows for interactive design through a TCP/IP network connection to a real or simulated robot, over which various robot poses can be communicated to or from the robot and manipulated locally using the KME graphical user interface. This portability and flexibility enables the user to work under different modes (configuration or cartesian space), with different robots (real or simulated), using different host machines (for running KME itself).

Pierris and Lagoudakis are with the Intelligent Systems Laboratory, Department of Electronic and Computer Engineering, Technical University of Crete, Chania, 73100, Greece gpierris@isc.tuc.gr, lagoudakis@intelligence.tuc.gr

KME was originally designed for and currently supports only the Aldebaran Nao humanoid robot (RoboCup edition v3) [1], which features a total of 21 degrees of freedom, and its simulated model on the Webots simulator [2]. However, the main features of KME can be easily configured for other robots and the tool itself could be used for a variety of purposes, such as providing complex motion patterns as starting points for learning algorithms and supporting educational activities in robot programming courses. KME has been employed successfully by *Kouretes*, the RoboCup team of the Technical University of Crete in Greece, for designing various special actions (stand-up, ball kicks, goal-keeper actions, bending of body), thanks to which the team ranked in the 3rd place at the RoboCup 2008 competition (Standard Platform League).

The remainder of the paper is organized as follows. Section II describes the Aldebaran Nao robot and Section III covers the features, functionality, and all technical aspects of KME. Section IV demonstrates a successful application of KME to the RoboCup competition, while Section V compares KME to a variety of similar motion editors, before discussing future work and concluding in Section VI.

II. THE ALDEBARAN NAO ROBOT

Nao is a 58 cm, 4.3 Kg humanoid robot (Figure 1) developed by Aldebaran Robotics based in Paris, France. Nao has not been released commercially yet, however Aldebaran's goal is to eventually promote Nao as an educational robotic platform and a family entertainment robot affordable to most budgets. The initial limited edition of the robot (RoboCup edition v2) made its debut at RoboCup 2008, as Nao was selected to be the official robot platform of the Standard Platform League.

The Nao robot carries a full computer on board with an x86 AMD Geode processor at 500 Mhz, 256 MB SDRAM, and 1 GB flash memory running an Embedded Linux distribution. It is powered by a 6-cell Lithium-Ion battery which provides about 45 minutes of continuous operation and communicates with remote computers via an IEE 802.11g wireless or a wired ethernet link. The Nao robot features a variery of sensors and actuators. Two 30fps, 640×480 color cameras are mounted on the head in vertical alignment, but only one is active at each time and the view can be switched from one to the other almost instantaneously. The two cameras provide non-overlapping views of the lower and distant frontal areas. A pair of microphones allows for stereo audio perception. Two ultrasound sensors on the chest allow Nao to sense obstacles in front of it and a rich intertial unit (a 2-axis gyroscope and a 3-axis accelerometer) in the torso

Fig. 1. The Aldebaran Nao Humanoid Robot (RoboCup Edition v3). [picture from www.aldebaran-robotics.com]

provides real-time information about its instantaneous body movements. Finally, an array of force sensitive resistors on each foot delivers feedback on the forces aplied to the feet, while encoders on all servos record the actual joint position at each time and two bumpers on the feet provide information on collisions of the feet with obstacles. The Nao robot has a total of 21 degrees of freedom; 4 in each arm, 5 in each leg, 2 in the head, and 1 in the pelvis (the 2 pelvis joints are coupled on one servo and cannot move independently). Stereo loudspeakers and a series of LEDs complement its motion capabilities with auditory and visual actions.

The Nao programming environment is based on the proprietary Naoqi framework which serves as a middleware between the robot and high-level languages, such as C, C++, and Python. NaoQi offers a distributed programming and debugging environment which can run embedded on the robot or remotely on a computer and offers an abstraction for event-based parallel and sequential execution. Its architecture is based on modules and brokers which can be executed onboard the robot or remotely and allows the seamless integration of various heterogeneous components, including proprietary and custom-made functionality. A simple, higher-level, user-friendly programming environment is also provided by the recently released proprietary Choregraphe software, which is further discussed in Section V due to its similarity to KME. Finally, there are realistic computer models of the Nao robot available for both the Webots robot simulator and the Microsoft Robotics Developer Studio.

III. KOURETES MOTION EDITOR

A. KME Concept

The goal behind the development of KME is to provide an abstract motion design environment for the common robot practicioner, which hides away the technical details of low-level joint control and strikes a balance between formal motion definition using precise joint angles in the configuration space of the robot and intuitive motion definition using manual joint positioning in the real-world work space of the robot. Such an abstraction yields a number of benefits, which served also as the driving force behind this work: (a) arbitrarily complex motion patterns can be easily designed without ever writing a single line of code, (b) motion patterns can be rapidly designed and tested through a simple and friendly interface, (c) motion patterns designed by one user can be easily shared, understood, used, and modified by other users, (d) various real and/or simulated robots can be accommodated simply by reconfiguring the back-end of the tool, (e) resulting motion patterns can be used as seeds in learning algorithms for further fine-tuning, and (f) proprietary motion patterns could be reverse-engineered as recorded sequences of complete or partial robot poses and subsequently manipulated at will.

Towards this end, KME is implemented as a client-server architecture, whereby the client and the server sides are interconnected over a TCP/IP network as described below. The server is a special "controller" attached to the real or the simulated robot, in the sense that it is platform-dependent, resides either on the robot or on some machine connected to the robot, and has the ability to directly control the robot joints. The server simply listens for a client on specific ports and undertakes the role of transferring joint values between the robot and the client, once a client is connected. The client is an independent software application running on the local or any remote machine and provides the graphical user interface (GUI) described below. Communication between the client and the server is bi-directional; any set of joint values provided by the client can be transferred to the server and drive the robot joints to the designated pose, and conversely the current joint values on the robot can be read and transferred from the server to the client for storage and further manipulation. Note that other platform-dependent information, such as sensor readings, can be communicated between the client and the server, if necessary, with appropriate adaptation of both the client and the server. Also, several clients can be connected to and/or disconnected from a single server to facilitate access to the robot from multiple KME users, mostly as a means of enabling quick user switching and coping with network failures, and less for simultaneous motion editing, which is certainly not recommended.

For maximum portability, it was decided to use open source software for the development of KME. In particular, the core client code is written in C++ and can be compiled under any popular operating system (Linux, Windows, MacOS) using a standard C++ compiler. The current graphical environment is based on FLTK libraries [3], however the core client code is structured in a way that allows interfacing with other popular graphical toolkits, such as QT and Tcl/Tk. Finally, the server code for the Nao robot has been written in the interpreted script language Python for simplicity and portability. Our preferred execution mode is to run the server on a remote machine which communicates with the Nao robot using the distributed NaoQi architecture.

B. Networking

All interprocess communication between server and client is based on the TCP/IP protocol. Once started, a server looks for an available port beginning from port 50000 and increasing the port number by 1 until an available port is found or a predefined limit is exhausted. If a port is found, the server listens for clients connecting to that port. When

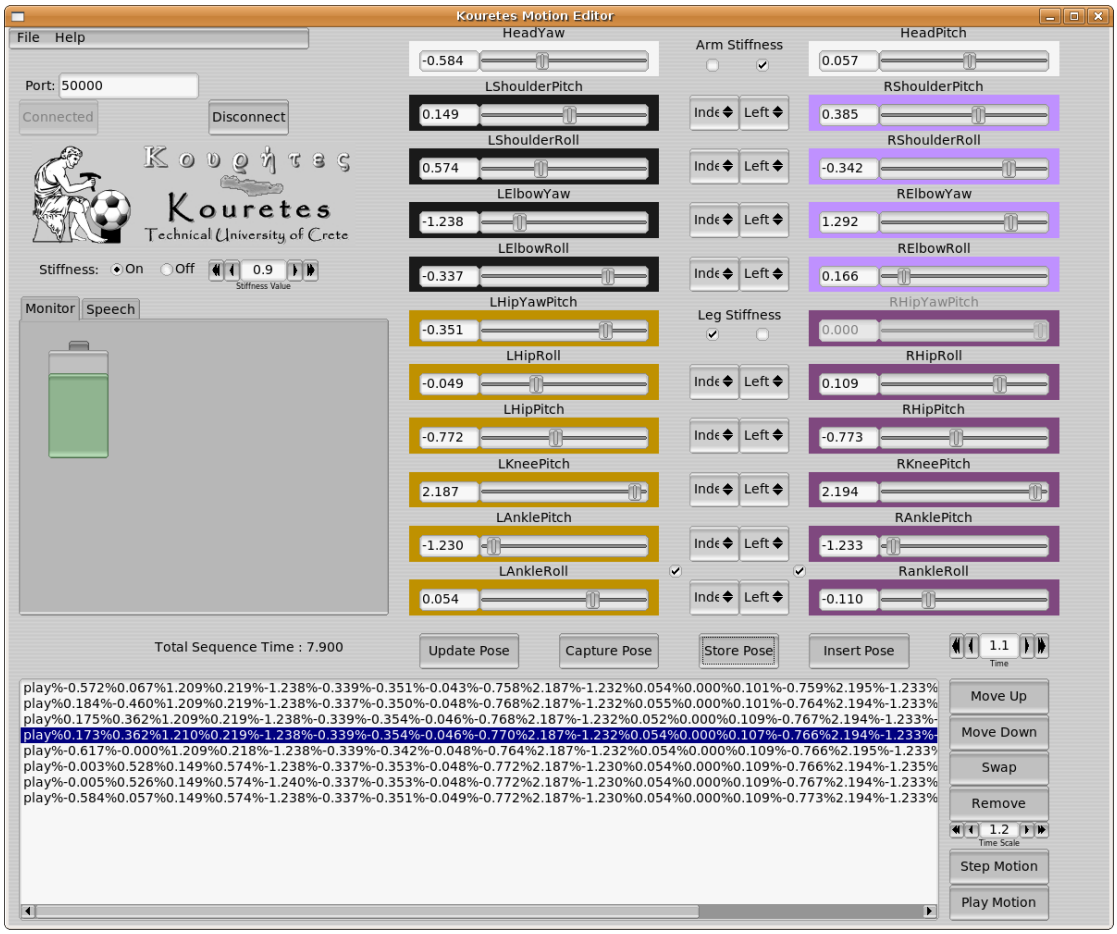

Fig. 2. Graphical user interface of the Kouretes Motion Editor.

started, the client must use the port number posted by the server to establish a connection. All messages exchanged over the network use simple ASCII-based structure for maximum portability purposes. Most messages contain complete robot poses, but there are also other smaller messages, for example, for initializing the connection, setting joint stiffness values, or communicating optional sensor information. The largest message communicated between client and server has a size of 144 bytes and corresponds to a complete robot pose description consisting of 22 signed floating-point numbers with 5 or 6 ASCII characters per number ([−]x.xxx format) separated by a marker (%). Messages are exchanged over the network only as needed; the client talks to the server only when the robot pose set by the client changes and symmetrically the server talks to the client only when the current pose on the server side is requested by the client.

It is quite important to understand that communication must meet real-time constraints given that any change on the client side must be immediatelly reflected at the robot joints on the server side. The requirement of minimized latency is dictated not only by efficiency goals, but also by safety concerns; a motion executed on the robot several seconds or minutes later than expected due to a network freeze or congestion may have fatal consequences. Under the current communication protocol, the server and the client run smoothly and transparently in a robust and predictable way even over the wireless link between the robot and the computer.

C. The Graphical User Interface

The KME graphical user interface (GUI) provides the means for the creation, capture, management, storage, reproduction, and export of any sequence of robot poses. The entire GUI consists of three components as shown in Figure 2. The component on the left-hand side contains the menu for file operations, the form and buttons for establishing a connection to a server using a specific port number, the radio button for turning joint stiffness on and off, and a tabular display for visualizing optional platform-specific sensor information. The component on the right-hand side offers 22 sliders, one for each joint of the robot; two of them (L/R HipYawPitch) are coupled together by default. Notice that the layout of the sliders resembles the location of the joints on the Nao robot and each joint chain (arm or leg) is marked by a distinct color. The user can set the value of any joint either by sliding the corresponding slider to the desired position or by setting directly the desired arithmetic value. Any change made to a joint value is immediately communicated to the robot through the server (if stiffness on the robot joints

is enabled), therefore the current robot pose coded by the slider values is always reflected on the robot. Symmetric joints, for example LShoulderPitch and RShoulderPitch, can be optionally coupled using the corresponding drop-down menus to select the type of coupling (matching or mirrored configuration) and the dominant joint (left or right). This feature is useful for creating symmetric or anti-symmetric motion patterns. A series of check buttons can be used to turn on/off stiffness locally on specific joint chains to allow design of motion patterns using only a single joint chain. Finally, the component at the bottom is a robot pose sequence editor. The sequence of poses can be edited as needed; poses can be inserted to or deleted from the sequence, they can be swapped with other poses, and they can be moved up and down to the desired place in the sequence. The KME GUI can be customized for any robotic platform using an `.xml` configuration file which contains the total number of joints, a unique name for each joint, a maximum and a minimum value for each joint, the step size of value changes for each joint, a color for the slider of each joint, and the offered couplings between pairs of joints.

D. Motion Design

Any complex motion pattern created using KME is a timed sequence of robot poses in the configuration space. Robot poses can be created either by setting joint values through the sliders of the GUI or by capturing the current joint values of the real or simulated robot. The current robot pose coded in the sliders can be captured and stored at the end of the sequence using the "Store Pose" button or right after the currently selected position using the "Insert Pose" button . The "Update Pose" button can be used to update the currently selected pose with new values. Alternatively, for pose generation the user may manually move the robot joints to any desired configuration (under no stiffness) and use the "Capture Pose" button to capture the current joint values. The user can also adjust the transition time between subsequent poses, which implicitly determines the speed in the motion of each joint. The time value stored with each pose is the transition time from the previous pose to the current one. Finally, the user can "play" the current pose sequence from any point (either in a step-by-step fashion or continuously) to observe the complete motion pattern on the robot. Once the desired movement is complete, the pose sequence (or its symmetric one according to the sagittal plane of the robot body) can be exported to a file and can be further used within any robot controller by simply invoking a motion execution routine.

Designing motion patterns using KME can become a lot more interactive, as we discovered along the way. Consider two consequent robot poses A and B. The user may want the robot to move from pose A to pose B in a certain amount of time, however this may not be possible because of limited acceleration or mechanical load constraints. Using the step-by-step execution, the user may try to play the pose transition from A to B, however the robot may end up in some other pose C under insufficient time or under mechanical load stress on the servos. Instead of trying to fix pose B (or A), the idea is to capture pose C from the current joint values; the transition from A to C is clearly a safe one. The design of the remaining motion pattern begins now from C and may shoot either for B (if possible) or for another target robot pose. Building the motion pattern in this iterative manner yields a motion sequence which complies with time and load constraints.

E. Motion Execution

Motion patterns designed using KME can be subsequently incorporated and reproduced within any robot controller without requiring the presence of KME itself. This is accomplished using a simple C++ routine which simply executes the stored motion patterns. In particular, the KME files found on board the robot are loaded into the main memory during initialization. A call for motion execution specifies the name of the desired motion pattern and a time-scaling factor (a real number around 1.0). The executor routine retrieves the specified pose sequence and executes the poses sequentially using a linear interpolation between them to avoid motion jerkiness. The time-scaling factor is used for speeding up or slowing down the execution as it multiplies the time values of each pose. A value of 1.0 corresponds to the nominal stored execution time; the user may optionally export the scaled or the nominal time values in the motion file. It should be noted that the motion executor is a simple open-loop scheduler; unexpected and/or uncertain events should be handled by a higher-level behavior module.

IV. EMPIRICAL EVALUATION

A. The RoboCup Competition

In its short history, the RoboCup competition [4] has grown to a well-established annual event bringing together the best robotics researchers from all over the world. The initial conception by Hiroaki Kitano in 1993 led to the formation of the RoboCup Federation with a bold vision: "*By the year 2050, to develop a team of fully autonomous humanoid robots that can win against the human world soccer champions*". The uniqueness of RoboCup stems from the real-world challenge it poses, whereby the core problems of robotics (perception, cognition, action, coordination) must be addressed simultaneously under real-time constraints. The proposed solutions are tested on a common benchmark environment through soccer games in various leagues,with the goal of promoting the best approaches, and ultimately advancing the state-of-the-art in the area. Beyond soccer, RoboCup now includes also competitions in search-and-rescue missions (RoboRescue), homekeeping tasks (RoboCup@Home), robotic performances (RoboDance), and simplified soccer leagues for K-12 students (RoboCup Junior).

B. The Standard Platform League

The Standard Platform League (SPL) [5] of the RoboCup competition (Figure 3) is among the most popular leagues, featuring two to four humanoid Aldebaran Nao robot players

Fig. 3. Standard Platform League at RoboCup 2008 in Suzhou, China.

in each team. This league was formerly known as the Four-Legged League featuring Sony Aibo robots which were replaced in 2008. Games take place in a $4m \times 6m$ field marked with thick white lines on a green carpet. The two colored goals (skyblue and yellow) also serve as landmarks for localizing the robots in the field. Each game consists of two 10-minute halves and teams switch colors and sides at halftime. There are several rules enforced by human referees during the game. For example, a player is punished with a 30-seconds removal from the field if he performs an illegal action, such as pushing an opponent for more than three seconds, grabbing the ball between his legs for more than three seconds, or entering his own goal area as a defender.

The main characteristic of the Standard Platform League is that no hardware changes are allowed; all teams use the exact same robotic hardware and differ only in terms of their software. This convention results to the league's characterization by a unique combination of features: autonomous player operation, vision-based perception, legged locomotion and action. Given that the underlying robotic hardware is common for all competing teams, research efforts have focused on developing more efficient algorithms and techniques for visual perception, active localization, omnidirectional motion, skill learning, and coordination strategies. During the course of the years one could easily notice a clear progress in all research directions.

C. KME at RoboCup 2008

KME was employed on Nao v2 by *Kouretes*, the RoboCup team of the Technical University of Crete, during the RoboCup 2008 competition which took place in Suzhou, China in July 2008. A much-needed motion in the SPL-Nao league is that of standing up after a fall. Using KME, it was fairly easy to design a stand-up motion pattern to recover from a fall (Figure 4) in reasonable time. It was discovered that the servos on the Nao v2 arms are quite weak to support its body weight, therefore one needs to carefully move most weight to the legs for a successful stand-up. It was also discovered that to recover from a face-up fall (or even a side fall), it was best to move first into a face-down pose and then execute the stand-up motion. The complete stand-up procedure uses the intertial sensors of the robot to determine the orientation of the robot body after a fall and executes appropriate motions that first bring the robot to a face-down pose before attempting a stand-up motion. Surpisingly, this stand-up motion designed on the carpet at the home laboratory did not work on the carpet at the competition venue. Thanks to KME, it took only about 30 minutes and two people holding the robot to design a new stand-up motion for the new carpet from scratch. KME was also used for designing other needed movements for the Nao league (bending of body, ball kicks, and goalkeeper actions).

The accompanying video file demonstrates the main features of KME and the motion patterns designed using KME and employed by Kouretes during the RoboCup 2008 competition. All clips from RoboCup 2008 were taken during actual games. Kouretes was the only one of the 15 participating teams that demonstrated a live stand-up motion and one of the few teams that used goalkeeper actions during the games. The team ended up winning the 3rd place in the league and a good deal of this success was due to KME.

V. RELATED WORK

KME offers some innovative ideas, however it also bears similarities to other existing tools for designing complex motion patterns. Choregraphe [6] is a proprietary software package developed by Aldebaran Robotics to facilitate complex behavior programming on the Nao robot. Its beta release came out in June 2008, well after the development of KME, but the functional release came out only in January 2009. Choregraphe offers a cross-platform environment that allows the user to build various movements and behaviors on a real, a simulated, or a VRML model of the Nao robot. To this end, it combines time-based and event-based approaches. Time-based design is used to schedule motions and multimedia material over time. Different timelines can be used depending on the current execution context. An event manager is responsible for identifying the current context based on the occurence of events and triggering the appropriate behaviors. The MEdit tool [7] was developed by Sony for the popular Aibo robot featuring a total of 20 degrees of freedom. MEdit is a graphical tool that allows users to generate complex motion patterns as sequences of poses. Unfortunately, its first (and only) release was incomplete in terms of functionality. While the design of motion patterns can be done through direct setting of joint values or through manual joint positioning on a VRML robot model, export and integration of such motion patterns into generic robot controllers and applications is rather cumbersome (each pose is saved as a separate file and the motion is uploaded to the robot only through the custom-made R-Tool and triggered only through the custom language R-Code). Skitter [8] is another motion editor for Aibo robots with more capabilities. It allows combinations of motions (20 degrees of freedom), lights (32 independent LEDs), and sounds (MIDI and WAV playback) along a time line. Such combinations are calleds skits and can last up to 4 minutes. Skits are designed using plots over time and a VRML robot model which can be manually positioned

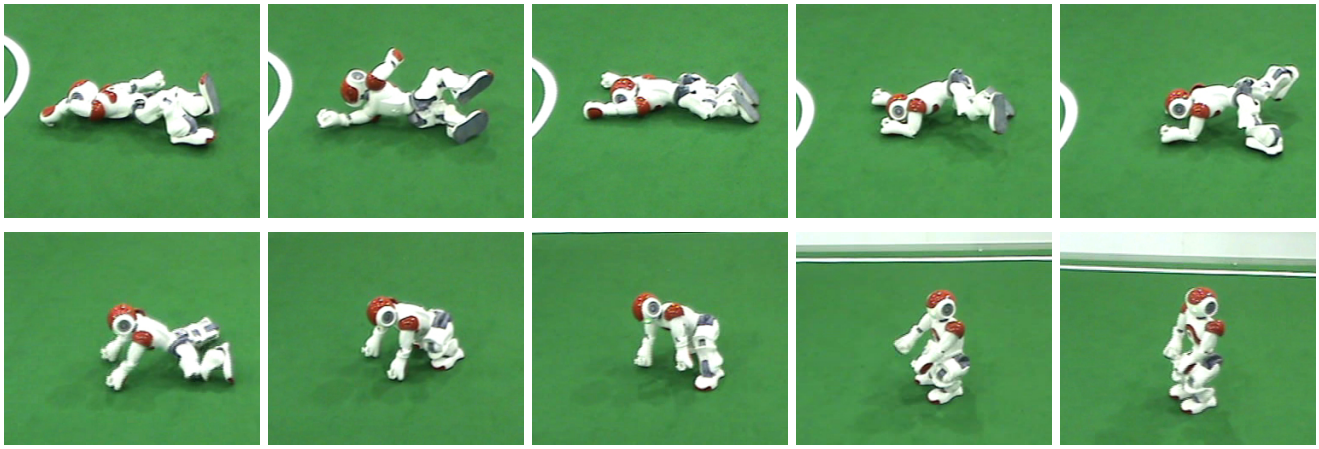

Fig. 4. Nao standing up from a face-up fall (demonstration at RoboCup 2008).

for capturing poses. The resulting motion patterns can be exported for use on the real robot, again only through R-Code. Finally, Motion Designer [9] is a tool developed by the RoboCup team Microsoft Hellhounds and comes in two versions, one for the Aibo and one for the Nao robot. Motion Designer allows the design of complex motions interactively with the real robot by offering several manipulation functions on timed sequences of poses and a custom transition editor for combining different motions.

Choregraphe's capabilities are well beyond KME capabilities, however it is not customizable and comes with heavy system requirements for reasonable real-time performance. MEdit and Skitter are tied only to the Aibo robot and do not support direct interaction with the real robot. MEdit, Skitter, and Motion Designer are available only for the Windows™ operating system. Apart from these differences, the distinguishing feature of KME missing from the tools mentioned above is the ability to directly interact in real-time with the real (or a realistically simulated) robot and design safe and robust motion patterns in an iterative manner, as well as the flexibility in incorporating the resulting motions in a variety of robot software architectures. It is important to stress out the fact that working directly with the real robot overcomes several difficulties occuring when motion patterns designed on a VRML model without sense of physics and/or mechanical, dynamic, and kinematic constraints do not yield the desired effect on the real robot. KME currently focuses only on motion, ignoring light and sound, which really fall outside its scope and purpose. In summary, KME complements the existing tools by providing an alternative, flexible, effective, interactive, and customizable motion-design tool, when motion alone is at focus.

VI. Future Work and Conclusion

Our short-term plan is to add several bells and whistles to the next release of the KME, such as support for partial configuration manipulation and/or execution over selected subsets of joints, motion safety and feasibility analysis, and constrained motion planning for interpolating between poses in the motion sequence using arbitrary criteria. In addition, we plan to release configuration files and server code for the Sony Aibo and the Robotis Bioloid robots. A major feature step would be to allow for manipulation of various control structures (events, loops, branches, sequences, etc.) over robot poses with the goal of designing simple closed-loop behaviors. Since robots are being increasingly popular in family entertainment, such tools will be undoubtedly important for the non-professional users.

This paper described the Kouretes Motion Editor (KME), an interactive software tool for designing complex robot motion patterns, freely available through **`www.kouretes.gr`**. While KME aims mostly for assisting RoboCup practicioners and robot educators, it can also be extremely useful in research efforts by simplying testing procedures and providing initial hand-crafted motions for various robot learning tasks. We believe that the development of such tools can also bridge the gap between humans and robots, as they abstract from the technical details, moving attention to the task at hand.

VII. Acknowledgments

The authors gratefully acknowledge the support of the administration of the Technical University of Crete to team *Kouretes*. This work was partially supported by the Marie Curie International Reintegration Grant MCIRG-CT-2006-044980 awarded to Michail G. Lagoudakis within the 6th European Framework Programme.

References

[1] D. Gouaillier and P. Blazevic, "A mechatronic platform, the Aldebaran robotics humanoid robot," *32nd IEEE Annual Conference on Industrial Electronics, IECON 2006*, pp. 4049–4053, November 2006.
[2] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004.
[3] B. Spitzak, M. Sweet, C. P. Earls, and M. Melcher, *The Fast Light Toolkit v. 1.3 Programming Manual*. [Online]. Available: www.fltk.org
[4] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, , and H. Matsubara, "Robocup: A challenge problem for AI," *AI Magazine*, vol. 18, no. 1, pp. 73–85, 1997.
[5] SPL Technical Committee, "RoboCup SPL (Nao) rule book," 2008.
[6] Aldebaran Robotics, *Choregraphe*. [Online]. Available: www.aldebaran-robotics.com/eng/choregraphe.php
[7] Sony Corporation, *MEdit for ERS-7 1.0.7, OPEN-R SDK*, 2003.
[8] *Skitter v3.40*. [Online]. Available: www.dogsbodynet.com/skitter.html
[9] Microsoft Hellhounds, *Team Report 2006*, Dortmund University, 2006.