Robotics and Autonomous Systems 57 (2009) 870–876

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Robot motion description and real-time management with the Harmonic Motion Description Protocol

Norbert Michael Mayer [a,*], Joschka Boedecker [b], Minoru Asada [b]

[a] Department of Electrical Engineering, National Chung Cheng University, 168 University Road, Min-Hsiung, Chia-Yi 62102, Taiwan

[b] JST ERATO Asada Project for Synergistic Intelligence and Emergent Robotics Laboratory, Graduate School of Engineering, Department of Mechanical Engineering, FRC-1, Osaka University, 2-1 Yamada-oka, Suita, Osaka 565-0871, Japan

ARTICLE INFO

Article history:
Available online 23 April 2009

Keywords:
Humanoid robots
Motion design
Harmonic functions

ABSTRACT

We describe the Harmonic Motion Description Protocol (HMDP), that can serve as a part in tools for rapid prototyping of behaviors in a hybrid simulation real robot environment. In particular, we are focusing on the RoboCup 3D Soccer Simulation League server that is currently evolving rapidly, and becomes a more and more useful open source, general purpose simulation environment for physical Multiagent systems. HMDP uses harmonic functions to describe motions. It allows for superposition of motions and is therefore capable of describing parametric motions. Thus, typical open loop motions (walking on spot, forward, turning, standing up) of small humanoid robots are readily available and can be optimized by hand. In conjunction with the HMDP some software tools and a small real-time motion generator (called Motion Machine) have been developed. The current implementation is very flexible to use and can easily be implemented in rather small embedded systems.

1. Introduction

Many developers of autonomous robot systems experience difficulties when designing a control system that is at the same time capable of high level sensor processing – in particular vision sensors – and motor control. This is particularly true for humanoid robots that can have around 20 degrees of freedom. For advanced systems (e.g. Honda's Asimo or Sony's Qrio robots) multitasking real-time OS systems (often with several CPUs) are available, that manage the entire sensor and actuator processing in real time. Low cost designs usually lack the sensor processing and are restricted to a remote controlled embedded CPU or even an analogue system (e.g. RoboSapiens, [1]). These systems completely lack autonomous behavior capabilities. For the medium to low cost designs (for example in RoboCup [2,3], where autonomous behaviors are demanded), the solution can be a hybrid design using 2 CPUs, one for motor control and one for sensor processing. On the one hand the sensor data processing – in particular vision – is done by a portable IBM-PC i386 like system with a custom broadband multitasking operating system (Windows, Linux, BSD) that does not have real-time capabilities (see for example [4–7]). The drivers for cameras and other devices are cheap and do not need further development. The motor control on the other hand is done by a micro-controller that performs pre-defined motion patterns. The demanded motion pattern is communicated between both CPUs by a serial pipe or bus system, sometimes wireless. In particular, in humanoid robots, the motion control part has to be a real-time system. Motion patterns such as standing up need to be precisely executed in the range of a time-span of 100 ms, in order to produce a reliable performance. For this reason direct positional control from the PC side should be avoided.

During the development process of the robot's behavior problems arise from this hybrid design. Whereas a PC-like system is always accessible, ready for changes, the motor controller can only be accessed via specialized editors and development tools, debuggers etc. Changes of motor controller programs can only be realized by flashing the limited memory that is available on the controller board. The programming of the motor controller is mostly done in C by using many custom definitions that depend on the design features of the motor controller which vary in dependence of the product line and the manufacturer. Moreover, the real-time behavior is managed by a series of interrupts that are again dependent on the type of the controller. Thus, for prototyping, the development of motion patterns may turn out to be a bottleneck, and developers may be reluctant to change a working motion pattern. The problems can result in a development process in which the motion patterns are developed separately from the design of the overall behavior. This seems acceptable in robot systems that do not require a big set of motions and do not have many degrees of freedom.

* Corresponding author. Tel.: +886 5 2720411; fax: +886 5 2720862.
E-mail address: nmmayer@gmail.com (N.M. Mayer).

0921-8890/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2009.03.014

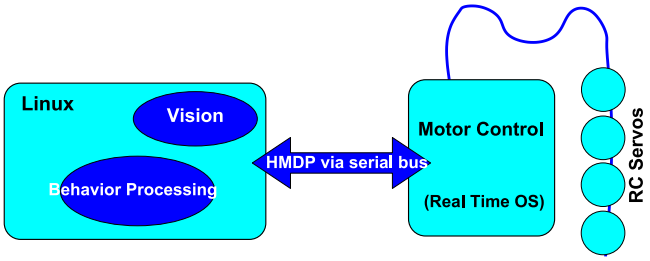

Fig. 1. Possible implementation of HMDP in a robot environment: Higher level behaviors are processed in a Linux micro-PC (e.g. Geode). The PC sends motion patterns over the serial bus to micro-controller. They are executed in real time.

In humanoid robots, however, this design principle is not really satisfactory. This is particularly true for soccer playing humanoid robots. Whereas humans have an infinite number of motion patterns available, the typical number of motion patterns of robots that participates, in the RoboCup competitions for instance is less than 10, e.g. strong kick, soft kick, walking, turn, several goal keeper behaviors. A first step would be to allow for the activation of several motion patterns at the same time. This can be used for looking for the ball and walking forward independently in parallel. Furthermore, it can be used to balance out perturbations from the walking process. Thus, in addition to the normal walking process a weak pattern can be added that can stabilize the walking motion. For this purpose it is necessary that the exact phase relation between both motion patterns is controllable.

This is one design policy for the protocol we present. The actual HMDP protocol defines the communication between the real-time motor controller and the PC-type vision- and behavior-processing computer—initially via serial bus (of course all kinds of other communication methods are possible, see also Fig. 1 for an illustration). The software environment which we describe in this work comprises also some tools for the motion design and handling, and a real-time motion generation and management system for the motion-controller CPU called *Motion Machine*. It is designed to be small and as possible hardware independent as possible.

In the robotics literature, some of the ideas that build the foundation of our protocol have occurred before. We find application of splines e.g. for trajectory generation of mobile robots (see [10] for the case of controlling an all-wheel steering robot, or [11] for an approach using a biped robot). Contrary to other works, though, we only store the Fourier coefficients of the motion splines and use these for motion generation in the micro-controller. We do not store any sampled spline curves in order to reduce communication load. The idea of control abstraction for a more compact representation (one of the design criteria for HMDP) of movements is realized with different methods, for instance Fourier analysis for cyclic motion patterns as in [12] or using hierarchical nonlinear principal component analysis to generate high-dimensional motions from low-dimensional input [13]. However, our main motivation for the creation of this protocol was not only to represent motions in a compact way, but also to address the issues concerning timing. Ideas related to the superposition principle of different motions can be found in [14]. There the pre-generated motions are analyzed and can be mixed together in frequency space (for instance to create smooth transitions). For this purpose an interpolation of Fourier coefficients is used. A powerful motion editor for small humanoid entertainment robots is described in [15]. In this work the created joint trajectories for the different motions are finally exported into files. These files contain control points and equations to interpolate between them to generate the desired motions in real time. Finally, a good overview of the general difficulties in motion planning for humanoid robots – specifically within RoboCup – is given in [16].

With regard to the simulation of robots similar problems arise from a different point of view. The RoboCup 3D Soccer Simulation League underwent some profound changes during the last several years. Currently realistic humanoid robots are simulated (see Fig. 3 for a screen shot of the NAO simulation in the official simulator called *Simspark* [17,9]). The motions of all robots are communicated between the agent program and the simulator as instantaneous updates of velocities of simulated servos communicated via TCP/IP. The role of HMDP in this framework can be to serve as an interface between the agent and the simulator (cf. Fig. 2). The details of the current state of the 3D Soccer Simulation League are discussed in publications that relate to the 3D2Real project [8,9]. The goal that is envisioned for the 3D2Real project is to have the finals of the soccer simulation league using real robots in the near future. For this ambitious goal several steps are necessary in the next years to create the necessary infrastructure and tools. According to the proposed road map in [8], a technical challenge would be held at the RoboCup competitions to test the ability to use the agent code of SSL participants on a predetermined real robot. We propose also the development of a *central parts repository* (CPR, see also [18]). This would be a collection of real robot designs, sensor and actuator models, complete robots, as well as controllers for certain architectures.

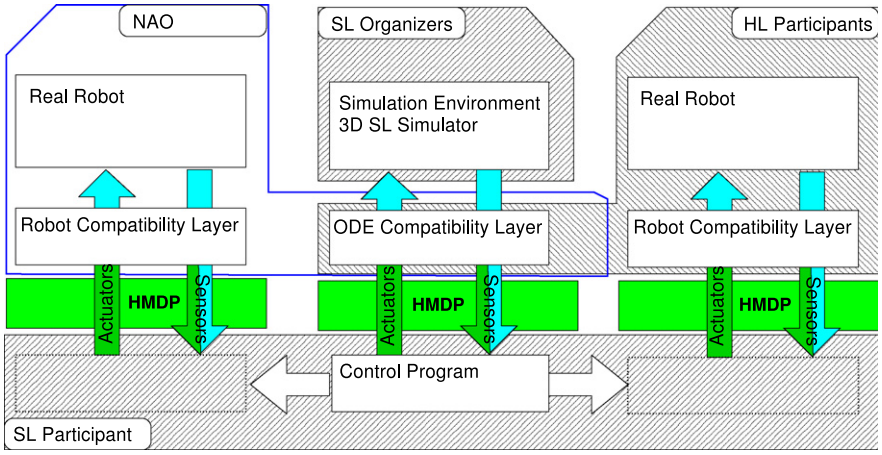

Fig. 2. Possible implementation of HMDP into a standardized software design and simulation environment (in this case 3D2Real [8]). The HMDP may serve at the same time as an interface for abstract motion description between agent and simulator as well as between agent and robot (see also [9]). The abbreviations stand for 3D Soccer Simulation League (SL), Soccer Humanoid League (HL); NAO is the name of the robot used in the new Standard Platform League. For details please see the above mentioned publications.

Fig. 3. A screen shot of the NAO soccer simulation in Simspark as used in the 3D Soccer Simulation League at RoboCup 2008.

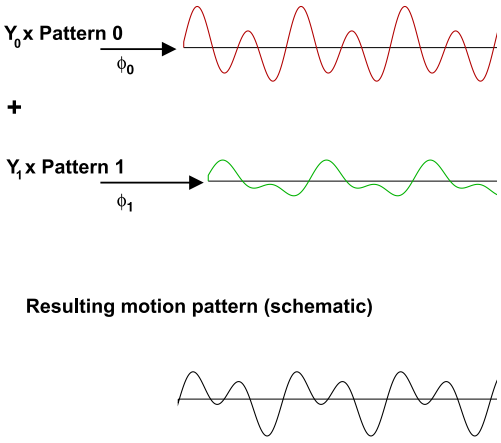

Fig. 4. Motion superposition: By using HMDP two or more motions can be superposed by defining the amplitudes Y_i and the phase shift ϕ_i. The resulting motion pattern is the sum of both initial patterns.

Participants of both Soccer Humanoid League (HL) and 3D Soccer Simulation League (SSL) would contribute to this repository according to their expertise and interest using a standard format.

The HMDP introduced in this work could be used as a standard for the motion description of the simulated and the real robots. In the simulation, the agents are connected to the simulator over the network. This means that they have to send messages (currently in ASCII strings) back to the simulator in order to specify, e.g., desired positions for motors. Since many agents connect to the simulator at once during a game this can lead to a high volume of network traffic that can cause high computational load on the server. If the HMDP was used for the description of motion patterns, longer messages describing the motions would only have to be sent sporadically when new patterns have to be set. Thus, the HMDP would provide a good solution for very related problems in the 3D soccer simulation league, and the humanoid league (as described in the introduction), and might eventually facilitate running the same code on simulated and real humanoids.

In the following section we outline the required specifications that follow from the motivations mentioned above. The syntax of the HMDP as implemented in our software is then briefly introduced. We describe the principle of superposition of motion, its advantages and potential problems. To illustrate the work with the protocol, we provide an example using an experimental graphical user interface. We also report our experiences using the protocol in a robot competition. Finally, we give an outlook on closed loop control using HMDP and a possible extended role of the HMDP inside the 3D2Real project and other projects, and close with a discussion.

2. HMDP specifications

The HMDP specifications are:

- The HMDP includes messages that are submitted from the PC (acting as master) to the micro-controller (acting as slave) and response messages from the micro-controller to the PC.
- The protocol allows for the PC side to set the current time as an integer and also to set the maximal time value after which the current time on the micro-controller is set to zero again.
- The protocol defines motion patterns in terms of splines. In order to allow for periodic motion patterns that can be repeated an arbitrary number of times the set of base functions is defined as a set of sines and cosines.
- The protocol activates motion patterns including the information at what time the motion pattern is activated, and its amplitude. It also defines which step of the motion pattern is assigned to what time step of the motion controller (motion phase assignment).
- The design of the HMDP includes the management of the motion patterns on the micro-controller side. It is possible to activate several motion patterns at the same time. The resulting motion pattern is the superposition (see Fig. 4) of all activated motion patterns (motion superposition principle).
- The protocol allows us to read out values of sensors that are connected to the micro-controller. In particular, it allows us to read out the the angle of the servo positions at a particular time step. The message for a sensor request consists of a vector of a time at which the sensor value should be read out and the name of the particular sensor. As soon as the time for read out is reached, the time value, the sensor name, and the sensor value are sent from the micro-controller to the PC.

3. Motion superposition

In this section we outline the principle with which motion patterns can be expressed in terms of motion splines; how they can be superimposed (cf. Fig. 4). The motion superposition is the one way to design a new parametric motion out of two existing motions. One key idea is to avoid to have any principle difference between non-parametric and parametric motions. It is possible to design all motions in a straightforward way for example in our motion designer (cf. Fig. 7).

In addition, we use harmonic functions, that is sine and cosine functions as a basis for our interpolation. Also, different sets of basis are applicable (e.g. cubic splines, other polynomial splines are possible). However, here we intended to use the same identical set of basis functions for both periodic (and eternal) motions (as mainly all possible types of walking) and non-periodic motions (stand up, kick, etc.). In this context of non-periodic motions the motion ends within the wavelength of one performance of the motion. If an appropriate set of cosine and sine amplitudes is chosen it is relatively simple to produce a smooth periodic motion.

In the following we discuss the harmonic motion splines for a robot with A actuators. Currently, a simple position control is implemented. We have spline functions that describe motion patterns $f_{p,a}(t)$. These are expressed in terms of discrete finite series of sine and cosine functions:

$$f_{p,a}(t) = c_0 + \sum_{0 \le n \le max} c_{2n+1,a} \sin(\rho\ \omega_{n,p}\ t) + c_{2n+2,a} \cos(\rho\ \omega_{n,p}\ t), \quad (1)$$

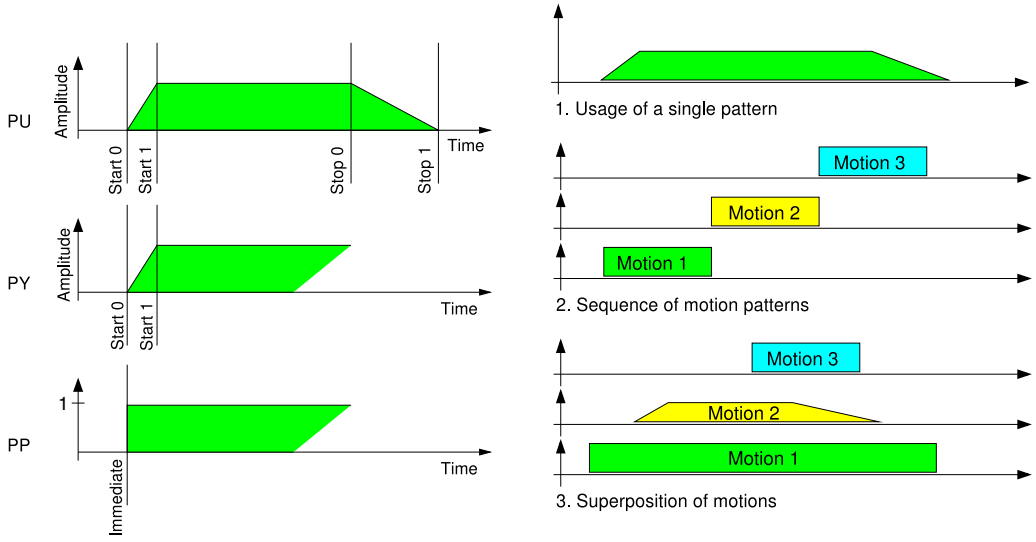

Fig. 5. Left: HMDP commands for different usages of the motion patterns. The PU command (for the HMDP syntax refer to [19]) can trigger a motion pattern with end. PY triggers the motion pattern without giving a point in time for the end. The PP command is for testing and starts a motion pattern immediately without end. Right: Examples for possible implementations of motion patterns. The third example is making use of the superposition principle.

where $\rho = \frac{\pi}{2N}, 0 \leq p < P$ is the index of the pattern and $0 \leq a < A$ the index of the actuator. The set wave numbers $\omega_{n,p}$ is specified when the pattern is initialized. The designer of the motion chooses the appropriate set of wave numbers – by hand in the GUI of the tool – in order to get the best interpolation of the particular set of postures. From the postures the coefficients c_i are calculated. As a convention of the current HMDP for a specific pattern p it is identical for all actuators a.

The vector $\mathbf{f}_p(t) = \{f_{1,p} \ldots f_{a,p}\}$ expresses then the state vector of the robot, i.e., the positions of all servos, given only the pattern p is active with an amplitude of 1. The final position that is sent to the servo is then:

$$\mathbf{F}(t) = \sum_{p<P} R_p(t)\mathbf{f}_p(t - \phi_p) \tag{2}$$

where the amplitude $R_p(t)$ and the offset ϕ_p are transmitted when the pattern is activated. Before the onset or change of the amplitude of a motion pattern the value of the offset between the motion onset and the time of the start of the motion ϕ_p, the – final – amplitude of the motion pattern $Y_{new,p}$, and the start times T_{start0}, T_{start1} have to be transmitted (see Fig. 5), $R_p(t)$ is then determined by

$$\begin{aligned} t < T_{start0} \quad &: R_p(t) = Y_{old,p} \\ t \in [T_{start0}, T_{start1}] \quad &: R_p(t) = (Y_{new,p} - Y_{old,p})/ \\ &\quad (T_{start1} - T_{start0}) \times t \\ t > T_{start1} \quad &: R_p(t) = Y_{new,p}. \end{aligned} \tag{3}$$

In other words the amplitude is changed in a linear way from the previous value to the current value. However, the value of ϕ_p changes at the time T_{start0}.

The total motion of the control output according to Eq. (2) is the superposition of all active motion patterns in their current amplitude. The virtues of this superposition might not be directly obvious in the general case. In the following we go into three different examples (see Fig. 5) where the superposition of motion patterns is useful.

First, examples for periodic movements: Independent movements concern disjoint sets of actuators and are applied by simply running both patterns at the same time. With respect to humanoid robotics this can be done by looking for the ball—that is: moving the head and walking at the same time. Both patterns can have different wave numbers and can be applied completely independent from each other. Here it is necessary that both movements do not interfere with each other, for example, the joint control signal should never collide under any circumstances. Also, it is only possible to have one pattern with dynamic effects on the whole body of the robot active at any time. In the case of walking and looking, only the walking would have an effect on the dynamic of the whole body of the robot. The dynamic effects of the looking for the ball should be negligible. This kind of combination is only possible if both movements concern completely independent joints and one movement pattern leaves the joint that concerns the other movement in a default position.

The second example would be two movements at the same frequency. Where the first movement is the default behavior and second movement is a response of the control to some perturbation. As an example, take vibrations during walking; these can be damped by adding a regulatory movement on top of the standard movement.

The third example would be parametric non-periodic movements, like kicking. Here, the kicking direction can be superposed to a standard kicking behavior. For a detailed description of these examples see Section 5.

4. Implementation details

The real-time motion generation and management system (Motion Machine, cf. Fig. 6) has been designed to be flexible with respect to the hardware used, and independent from external libraries. It uses its own self-defined float type, and can thus be used in systems that are initially restricted to integer operations. The Motion Machine is programmed in C. It consists of an input parser and the core real-time motion generator that calculates the control values as motion splines, as well as their superposition. Currently the Motion Machine is running on a Philips/Renesas ARM 7 architecture (CPU type LPC 2148), which includes 512 Kb FLASH memory and a total of 40 Kb SRAM. This type of micro-controller exceeds the demands of the Motion Machine by far. Table 1 gives an overview of the minimal demands and what is used in the current implementation.

In addition the HMDP syntax was designed to be relatively easy to parse.

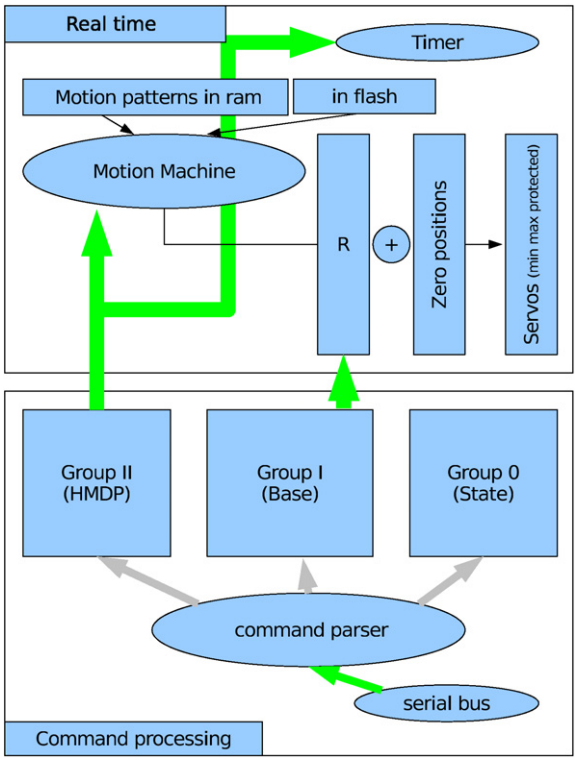

Fig. 6. Software architecture inside the motor controller (slave).

Table 1
Current implementation and expected minimal requirements for the Motion Machine.

	Current implementation	Minimal requirement
Program binary	26 kB FLASH	26 kB
Motion pattern	2 kB (fixed)	<2 kB[a]
Zero positions storage	2 kB	≈2 kB
Necessary RAM	12 kB	1–2 kB

[a] Depending on the number of coefficients.

The Motion Machine can be set into three distinct internal states. Depending on the activated state, different groups of commands have different effects. It is important to note that if a command is used in the wrong state the real-time property may be disrupted, the command may have an undesired effect, or no effect at all.

- State 0: The Motion Machine is de-activated and commands can be directed towards the motors. Command groups 0, I can be used.
- State 1: HMDP state. The Motion Machine is on and overwrites motor commands (group I). Instead, the HMDP commands that control the Motion Machine have to be used.
- State 2: "Plastic" state of the robot. The robot can be set manually into a state, and keeps the current posture, while changes in the current posture are possible by applying force to the servos. This is achieved by letting the servos always enforce the current position, that is read out at each timer interrupt. In this way, the robot is able to keep postures better than that in the state when the servos are turned off. The robot's joints are in a state that feels though still transformable like a medium of high viscosity. The viscosity of the joints can be altered by changing the gain of the servos. Fig. 7 is an example for a motion that was designed by using a sequence of postures which were modeled by using the state 2.

For designing motions a tool with a graphical interface (GUI) was used (*QMotion2* motion editor, depicted in Fig. 7). The widgets were designed by using the commercial cross-platform widget library Qt [21]. The motion patterns are first described as a sequence of postures from which the interpolation is calculated. The postures themselves can be defined in two different ways. First by moving sliders of the GUI, second by directly setting the robot into the plastic mode and this defining the posture by handling the robot (please confer [22] for a similar concept). The posture can then be read out. The second way proved to be economic for complicated movements like standing up. The motion editor further allows for copying and pasting of motion parts, changing states and changing the viscosity of the plastic state. In addition, two motions can be active at the same time at two different phase shifts, and thus the superposition of motions can be tested.

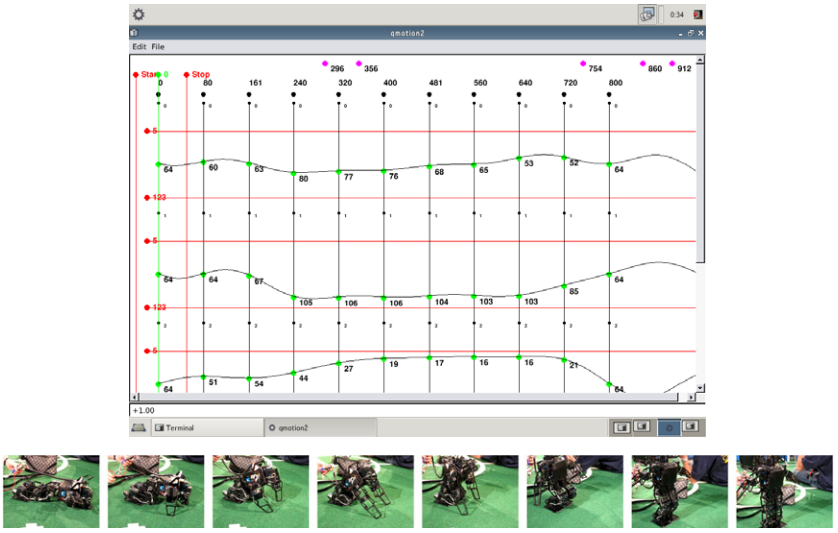

Fig. 7. Standing up motion designed by a tool to design HMDP motion patterns (screen shot). The screen shows in the top the activated frequencies of sines and cosine functions that perform the motion interpolation (dots in purple). Below the resulting interpolated motion functions for selected joints. The time increases from left to right. The first row depicts the left hip joint in roll direction, the second row the first left knee joint (robot type 4G [20] has 2 knee joints) and (half occluded) the motion of the second left knee joint. Below: The resulting motion for standing up of the robots (8 subsequent stages of the motion). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

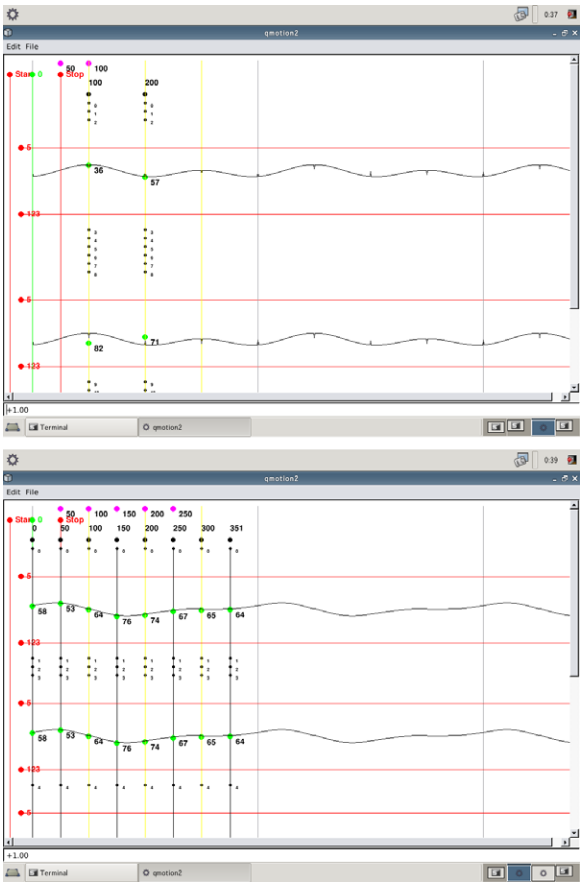

Fig. 8. Design of the walking motion which serves as a standard example for superposition as motions: The complete walking motion is designed from three motion patterns: Walking on spot (top), forward–backward (bottom) and a third motion that is turning right or left. All walking motions of the robot can then be achieved by an appropriate superposition of these three motions. For example a typical moving forward motion can be made by mixing the walking on spot motion with amplitude 1.0 with the forward–backward motion 0.05. Walking backward can be achieved by mixing walking on spot with forward–backward with a negative amplitude. In the case of backward walking better results can be achieved by adding some leaning forward (another very simple motion).

5. Results and applications using HMDP

The HMDP has been used for the Team JEAP robots [23] during the RoboCup 2007 competitions (for an overview of the RoboCup Humanoid League see [24]), and proved to be practically applicable there. All motions including walking in different ways, kicking, leaning forward, looking upward, sideward, standing up etc. could be designed in the described manner. The motions used in the competition were entirely open loop motions. However, closed loop approaches seem to be applicable in a useful way, see below. Fig. 7 shows a part of the motion design tool where the standing up motion as used during the RoboCup has been loaded. The standing up motion proved to be one of the few motions where the exact calibration is particularly essential. This might be caused by the fact that the arms of the robots of type 4G [20] are relatively short. The standing up motion is designed as a single motion and does not make use of the motion superposition principle. Whereas walking has been designed as the superposition of several motions mixed together in order to achieve walking in different directions (Fig. 8). The basic motion is walking on the spot. In addition, a forward motion that is to shift the standing leg and swinging leg backward and forward respectively. Finally, a turning motion. The final walking could be further improved by adding a bit of leaning forward to the walking motions. In this way the slip between feet and ground could be reduced.

6. Summary and outlook

We presented here an environment for the design and management of motions in small, medium priced humanoid robots. We assume for our system that the robots use the hybrid architecture of a non-real-time CPU for sensor processing and a real-time motion controller. Between these two CPUs we suggest a standardized protocol that can also be used in robot simulations for the communication between agent programs and the simulator. In addition, we presented a tool for motion design using the proposed protocol.

The advantage of the described method to previous designs is that in comparison to streaming of robot posture information the communication load can be reduced. At the same time motion design and management are still highly flexible (all kinds of parameter motions are possible). One important disadvantage to a standard control program on the motor controller is that closed loop approaches are more complicated to implement.

With regard to the outlook we focus in this section on two points: How closed loop control can be applied and on the role HMDP can serve in Simspark (RoboCup 3D soccer simulation environment. It is planned to add the code there). Fig. 9 illustrates both subjects. In the case of the closed loop control using HMDP one may design a compensation motion pattern for a specific expected perturbation (in the figure: healing pattern). At a specific phase of the motion pattern the Motion Machine can trigger a sensor reading. In the case of alarming sensor values the healing

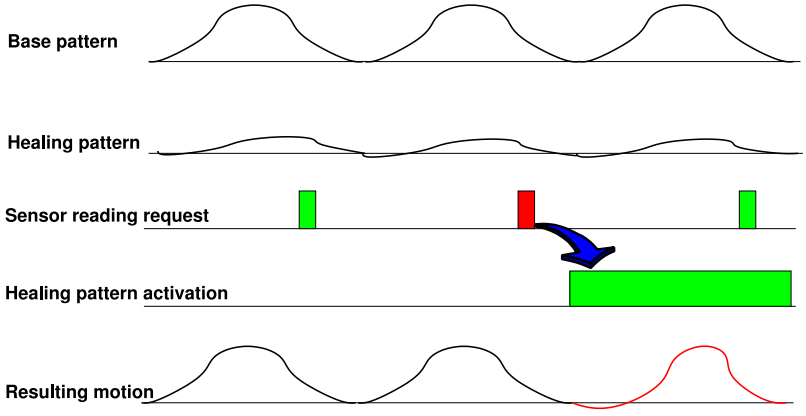

Fig. 9. Possible design of closed control loop. Although due to the necessary communication between the slave and master via HMDP and the consequent relatively large feedback delay the response (marked in red) can be set precisely in phase and fit together with the basic motion pattern (see Section 6). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

pattern can be activated in an appropriate amplitude. In this way although the feedback delay may be relatively large, the reaction precisely fits into the motion pattern.

The second point is that HMDP can serve as a standard interface to communicate motions. As an example we implemented it into the 3D soccer simulation server of the RoboCup competition. It may serve here to reduce the bandwidth of the communication between agents and the simulator. One reason for this idea is a problem with the communication load that is currently necessary between the agent and the simulation server during the game (the motion patterns themselves can be transferred before the game). A rough calculation of the load only for controlling motions gives a load of several hundred kilobits (22 actuators $\times$ around 10 bytes control code for each actuator command $\times$ the refresh rate).

From experiences with our real robot we know that it does not make sense to change motion patterns that drive the robot more than one time a second that would result in 50 bytes, one can achieve a significant reduction of the communication load, while the simulation is running, even if we assume that a motion pattern is changed 5 times a second.

Acknowledgements

We thank Dale Thomas for cross reading the article. We gratefully acknowledge the support of this work by the Japan Science and Technology Agency (JST), and a fellowship for young scientists from the Japan Society for the Promotion of Science (JSPS). N.M.M. thanks the Taiwanese National Science Council (NSC) and the Swiss National Fund (SNF) for their support. He also gratefully remembers the fruitful stay at Juergen Schmidhuber's laboratory at IDSIA, Lugano, Switzerland. He thanks all lab members for their warm hospitality, their interest and all the discussions there.

References

[1] Emanuele Menegatti, Alberto Pretto, Enrico Pagello, Artisti humanoid team for robocup 2007, 2007. Available online at http://jeap-res.ams.eng.osaka-u.ac.jp/~michael/quali/.
[2] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, H. Matsubara, RoboCup: A challenge AI problem, AI Magazine (1997).
[3] H. Kitano, M. Asada, The Robocup humanoid challenge as the millennium challenge for advanced robotics, Advanced Robotics 13 (8) (2000) 723–736.
[4] Sven Behnke, Michael Schreiber, Jörg Stückler, Hauke Strasdat, Konrad Meier, Nimbro kidsize 2007 team description, in: RoboCup Symposium 2007 CD-Rom, 2007.
[5] Jacky Baltes, Shane Yanke, University of manitoba humanoid robot team, in: RoboCup Symposium 2007 CD-Rom, 2007.
[6] Martin Friedmann, Jutta Kiener, Sebastian Petters, Dirk Thomas, Oskar von Stryk, Darmstadt dribblers, team description for humanoid kidsize league of robocup 2007, in: RoboCup Symposium 2007 CD-Rom, 2007.
[7] Carlos Antonio Acosta Calderon, Changjiu Zhou, Pik Kong Yue, Mike Wong, Mohan Rajesh Elara, Guohua Yu, Chin Keong Ang, Bi Wu, Robo-erectus junior, a kidsize soccer playing humanoid robot, in: RoboCup Symposium 2007 CD-Rom, 2007.
[8] Norbert Michael Mayer, Joschka Boedecker, Rodrigo da Silva Guerra, Oliver Obst, Minoru Asada, 3D2Real: Simulation league finals in real robots, in: RoboCup 2006: Robot Soccer World Cup X, in: Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, Tomoichi Takahashi (Eds.), Lecture Notes in Computer Science, Springer, 2007, pp. 25–34 (Paperback).
[9] Norbert Michael Mayer, Joschka Boedecker, Minoru Asada, On standardization in the robocup humanoid leagues, in: Proceedings of the Send Workshop on Humanoid Soccer Robots at HUMANOIDS 2007 (CD-ROM), 2007. Available online at http://www.informatik.uni-freiburg.de/~rc06hl/ws07/.
[10] Thomas Howard, Alonzo Kelly, Trajectory and spline generation for all-wheel steering mobile robots, in: Proceedings of the the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'06, October 2006, pp. 4827–4832.
[11] G. Bessonnet, P. Seguin, P. Sardain, A parametric optimization approach to walking pattern synthesis, International Journal of Robotics Research 24 (7) (2005) 523–536.
[12] Henning Schmidt, Dieter Sorowka, Frank Piorko, Negib Marhoul, Rolf Bernhardt, Control system for a robotic walking simulator, in: Proceedings of the 2004 IEEE International Conference on Robotics and Automation, 2004.
[13] Koji Tatani, Yoshihiko Nakamura, Reductive mapping for sequential patterns of humanoid body motion, in: Proceedings of the 2nd International Symposium on Adaptive Motion of Animals and Machines, 2003.
[14] Jan Hoffmann, Uwe Düffert, Frequency space representation of transitions of quadruped robot gaits, in: Computer Science 2004, Twenty-Seventh Australasian Computer Science Conference, ASCS2004, vol. 26, 2004, pp. 275–278.
[15] Y. Kuroki, B. Blank, T. Mikami, P. Mayeux, R. Playter, K. Nagasaka, M. Raibert, M. Nagano, J. Yamaguchi, Motion creating system for a small biped entertainment robot, in: Proceedings of the the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'03, vol. 2, 2004, pp. 1394–1399.
[16] Stefano Carpin, Enrico Pagello, The challenge of motion planning for humanoid robots playing soccer, in: Proceedings of the Workshop on Humanoid Soccer Robots of the 2006 IEEE-RAS International Conference on Humanoid Robots, 2006, pp. 71–77.
[17] Oliver Obst, Markus Rollmann, SPARK — A generic simulator for physical multiagent simulations, Computer Systems Science and Engineering 20 (5) (2005).
[18] Joschka Boedecker, Norbert Michael Mayer, Masaki Ogino, Rodrigo da Silva Guerra, Masaki Kikuchi, Minoru Asada, Getting closer: How simulation and humanoid league can benefit from each other, in: Kazuyuki Murase, Kosuke Sekiyama, Naoyuki Kubota, Tomohide Naniwa, Joaquin Sitte (Eds.), Proceedings of the 3rd International Symposium on Autonomous Minirobots for Research and Edutainment, Springer, 2005.
[19] Norbert Michael Mayer, HMDP technical description, 2007. http://www.jeap-res.ams.eng.osaka-u.ac.jp/~michael/hmdp.
[20] Hitoshi Takayama, Reo Matsumura, Naoki Shibatani, Takuro Imagawa, Takeshi Maeda, Takahiro Miyashita, Tomotaka Takahashi, Yohei Akazawa, Nobuo Yamato, Hiroshi Ishiguro, Team Osaka A (kid size) team description paper, in: RoboCup 2006 Symposium Papers and Team Description Papers CD-ROM, 2006.
[21] Oslo Trolltech A.S.A., Qt: Cross-platform rich client development framework, 2008. http://trolltech.com/products/qt.
[22] Fabio Dalla Libera, Takashi Minato, Ian Fasel, Hiroshi Ishiguro, Emanuele Menegattie, Enrico Pagello, Teaching by touching : An intuitive method for development of humanoid robot motions, in: 2007 IEEE-RAS International Conference on Humanoid Robots, 2007.
[23] Norbert Michael Mayer, Masaki Ogino, Rodrigo da Silva Guerra, Joschka Boedecker, Sawa Fuke, Ayako Watanabe, Kazuhiro Masui, T. akanori Nagura, Tomomi Ooide, Minoru Asada, JEAP team description, RoboCup Symposium 2007 CD-Rom, 2007.
[24] Norbert Michael Mayer, Minoru Asada, RoboCup humanoid challenge, International Journal of Humanoid Robotics 5 (2008) 335–351.

Norbert Michael Mayer studied physics in Frankfurt am Main and Goettingen, Germany. He received his Ph.D. in physics at the Max Planck Institute for Dynamics and Self-Organization. In 2001 he developed software for 3D scanning systems and finally turned towards robotics. From 2002 to 2004 he worked as a group leader at the GMD Japan Research Laboratory in Kitakyushu, Japan, where he was responsible for the RoboCup activities. Between 2004 and 2009 he worked at the Osaka University in the JST ERATO Asada Project for Synergistic Intelligence. Since then he is an Associate Professor at the National Chung Cheng University in Chia-Yi County, Taiwan.

Joschka Boedecker received his diploma in computer science from the University of Koblenz-Landau, Germany, in 2006. He spent one year in 2002/2003 at the AI Department of University of Georgia, USA, on a scholarship from the German Academic Exchange Service (DAAD). In 2005, he stayed at the Department of Adaptive Machine Systems of the Osaka University, Japan, for half a year as a visiting researcher under the support of a scholarship from the Heinz Nixdorf Foundation. Since 2006, he is a Ph.D. student at the same department, supported by a JSPS Fellowship for Young Scientists. His research interests include dynamics shaping and information dynamics in recurrent neural networks, and application of reservoir computing for robot control. He is a graduate student member of IEEE.

Minoru Asada received the B.E., M.E., and Ph.D., degrees in control engineering from the Osaka University, Osaka, Japan, in 1977, 1979, and 1982, respectively. From 1982 to 1988, he was a Research Associate of Control Engineering, Osaka University, Toyonaka, Osaka, Japan. April 1989, he became an Associate Professor of Mechanical Engineering for Computer-Controlled Machinery, Osaka University, Suita, Osaka, Japan. April 1995 he became a Professor of the same department. Since April 1997, he has been a Professor of the department of Adaptive Machine Systems at the same university. From August 1986 to October 1987, he was a visiting researcher of Center for Automation Research, University of Maryland, College Park, MD. Since 2002, he has been the president of the International RoboCup Federation. Since 2005, he has also been the research director of "ASADA Synergistic Intelligence Project" of ERATO (Exploratory Research for Advanced Technology by Japan Science and Technology Agency).