BEYOND THE LIMITS OF KINEMATICS IN PLANNING KEYFRAMED BIPED LOCOMOTION

Tamás Juhász[1,2], Tamás Urbancsek[2]

[1] Department Virtual Engineering, Fraunhofer Institute for Factory Operation and Automation, Sandtorstrasse 22, 39106 Magdeburg, Germany
[2] Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Magyar tudósok krt. 2, 1117 Budapest, Hungary

Tamas.Juhasz@iff.fraunhofer.de (Tamás Juhász)

Abstract

The methods of locomotion planning for biped robots have been studied for many years. Presently, several companies have announced the commercial availability of various humanoid robot prototypes with diverse controllability.
However a family of existing humanoids is being controlled using so called “keyframes”. The keyframe technique lets us specify the target angular position of the joints only in discrete time-steps, whereby the length of the individual intervals is also defined. These platforms have built-in path planning algorithms and have their own control electronics: i.e. no external feedback is given about actual joint torques. The widely spread locomotion planning methods cannot be applied directly to this kind of robots, thus they need a different approach.
In this paper, we present an iterative technique of keyframed humanoid motion planning based on numerical optimization methods for the application of stable biped stair climbing.
Using inverse kinematics and the existing techniques for creating keyframed character animation, one can relatively easily generate a reference movement, which can be executed real-time using a fast kinematics model of the given robot. In this case we get the reference time curves of the robot segments’ poses. It is a fact, that due to gravitational-, inertial- and frictional effects, the motion won’t exactly be the same in the real world. Assuming we have built a detailed, sophisticated dynamics model (running offline) we can formulate a norm that describes the distance between these outputs. In this article we present our idea, how the motion could be automatically tailored by lowering this norm using numerical methods in a way, that the output of the dynamic model better approximates the reference motion.
Finally, we show our experimental results obtained by implementations running within a modern simulation environment as well as on our test humanoid platform.
This paper is an extended and revised version of [1].

Keywords: Humanoid, Keyframe, Motion planning, Dynamics, Stair climbing

Presenting Author’s biography

Tamás Juhász is an early stage researcher at the Virtual Engineering Expert Group of Fraunhofer Institute in Magdeburg, Germany. Since 2003 – being a PhD student of informatics at BUTE, Hungary – he has published many lectured articles in the field of mobile robot simulation and advanced 3D visualization techniques.

1 Introduction

Humanoids should keep their dynamic balance in order not to fall down while walking. Therefore besides geometric motion planning and kinematics, dynamical effects should also be taken into account.

In case of a biped locomotion, start and goal positions are given in a virtual environment. Some existing motion-planner methods (the lazy RPM procedure [2], the randomized planning techniques [3][5] or the footstep planning [4]) give continuous time functions how to move the robot through the desired path. This means, the reference signal for all the robot joints will be set in every moment.

Albeit, these techniques cannot be used directly for those kinds of systems, which can only be regulated in a discrete time manner externally (using so called "keyframe" inputs). Keyframed humanoids allow the user to specify target servo angles only for discrete moments; their onboard path planner interpolates the reference path for the actual time using n^{th} order polynomial approximations.

In a special case, when $n = 1$, we talk about linear interpolation: the reference path in servo space is a broken line; the reference (angular) speed of the every (rotational) servo remains constant within each of the intervals.

The joint correction signals are mostly produced in a decoupled way by on-board or in-servo controllers and the user has no access to them.

Some systems do not even allow real-time tracking of motion execution; actual joint values cannot be read externally. For these robots the whole motion shall be planned in advance.

This paper deals with these biped platforms and proposes a different approach for locomotion planning.

2 Keyframe based motion planning

2.1 Keyframed character animation

Among the various numerical techniques, the keyframe methodology can also be used to define a robot motion.

Each keyframe includes a snapshot of robot servo parameters and a timestamp that defines the time of sampling. Thereby a robot motion can be defined by a series of keyframes – hereinafter we will refer to them also as motion phases.

2.2 Role of inverse kinematics and inverse geometry in motion planning

The keyframe-based method combined with inverse kinematics makes character animation design faster. The animator defines the new motion phase by altering the previous one. He can move segments both in servo and in Cartesian space: e.g. lifting up left foot or translating body forward. Servo values are computed then by the inverse geometric model. If the animator takes a snapshot of the actual robot state a new motion phase is created.

Using this technique an arbitrary locomotion can be designed. The reference motion is played with help of the kinematic model.

2.3 Kinematic model

Let us assume our biped robot has J joints, we will refer them to $1 \le j \le J$. The robot has then $(J+1)$ segments indexed by $0 \le j \le J$. We can define K keyframes, when we define the τ_k length of the k^{th} time interval ($\rightarrow \boldsymbol{\tau}$ vector) between the phases k and $(k-1)$, as well as the q_{kj} joint angles ($\rightarrow \boldsymbol{\Theta}^{[K \times J]}$ matrix).

The robot has a hierarchical structure, where the torso segment plays the role of the root node. If we know all joint angles, we can calculate the local pose of each segment (where "local" means relative to the torso).

Let us call the initial pose of the torso segment ${}^{0}\boldsymbol{\Gamma}_{0}$ which is a 6 component vector in global Cartesian space as 3 position coordinates and 3 Euler-angles describe the pose for each segment. The upper left index denotes the base coordinate system of the humanoid attached to the robot torso, the right index refers to the initial state at $t = 0$.

In kinematical modeling it can be assumed that the pose of a single robot segment does not change during the motion phase. This so called "unyielding" object is always one of the feet: it stays on the ground and every segment moves relatively to it during the given phase. The corresponding objects' references are stored in the $\mathbf{u}$ vector with length k.

If we know the interpolation method between the keyframes, we can formulate a continuous vector-functional, describing the pose of the j^{th} segment in global Cartesian-space:

$$ {}^{j}\boldsymbol{\Gamma}^{Kin}(\boldsymbol{\Theta}, \mathbf{u}, \boldsymbol{\tau}, {}^{0}\boldsymbol{\Gamma}_{0}, t), 0 < j \le J \qquad (1)$$

The ${}^{j}\boldsymbol{\Gamma}^{Kin}$ vector has a dimension of 6 (position x,y,z + Euler-angles φ,υ,ψ). $\boldsymbol{\Gamma}^{Kin}$ denotes the aggregate pose vector with length of $6*(J+1)$, representing the time-dependent pose of all segments:

$$\boldsymbol{\Gamma}^{Kin}(\boldsymbol{\Theta}, \mathbf{u}, \boldsymbol{\tau}, {}^{0}\boldsymbol{\Gamma}_{0}, t) \qquad (2)$$

From now on, the $\boldsymbol{\Gamma}^{Kin}$ functional will be called the kinematic model of the robot.

3 Iterative motion correction using dynamics model

Let us assume we have a well designed reference motion using the keyframe-technique: apparently this will run only in the (ideal) virtual world smoothly. Nevertheless this motion represents exactly what we would like to achieve with the real robot (e.g.: stepping forward by a given step length). If the servos could exactly follow the reference signal, $\mathbf{\Gamma}^{Kin}(t)$ would describe the robot state in Cartesian space.

Considering the dynamic effects of the real world, there will be some difference between the real and the reference motions (e.g.: the given forward-step will be shorter or longer) – in extreme case the humanoid might also fall down.

This paper presents an algorithm that finds a new motion that approximates $\mathbf{\Gamma}^{Kin}$ better in real world.

3.1 Output of dynamics simulation

As a first step, the dynamic model of our robot has to be built (the general procedure is not included in this article, but details can be found in [7]).

Our motion-optimization process uses an iterative approach, where the $\mathbf{\Theta}$ input matrix and $\boldsymbol{\tau}$ will vary step by step. Using dynamic simulation we have the following output after i steps:

$$\mathbf{\Gamma}_{(i)}^{Dyn}(\mathbf{\Theta}_{(i)},\mathbf{u},\boldsymbol{\tau}_{(i)},{}^{0}\mathbf{\Gamma}_{0},t) \tag{3}$$

$\mathbf{\Gamma}_{(i)}^{Dyn}$ denotes a vector-vector function of dimension *6*(J+1)*, and it represents the pose of all the robot segments depending on time, but using the dynamic model for the particular time.

3.2 Iterative conformance enhancement

Taking a given iteration into account, the difference of the actual output of the dynamics simulation and the reference motion is also a vector-scalar function:

$$\mathbf{\Gamma}_{(i)} = \mathbf{\Gamma}_{(i)}^{Dyn} - \mathbf{\Gamma}^{Kin} \tag{4}$$

In some cases the position error has higher priority than the orientation error; furthermore it is usually desired to have better compliance on the feet as for example on the head. For these requirements we can define a

$$\mathbf{W} = diag < w_0, w_1, ..., w_J > \tag{5}$$

diagonal constant weight matrix with dimensions [(J+1)*6 × (J+1)*6]). We can define a weighted error function with help of inner product:

$$\mathrm{E}_{(i)}^{2}(t) = < \mathbf{W}\cdot\mathbf{\Gamma}_{(i)}, \mathbf{W}\cdot\mathbf{\Gamma}_{(i)} > \tag{6}$$

Using this error function we want to define a χ norm, in order to have a non-negative scalar value that represents the correspondence between the reference motion and the simulated one (considering the whole simulation length is T). At the ith step it computes

$$\chi_{(i)}\left\{\mathbf{\Theta}_{(i)},\mathbf{u},\boldsymbol{\tau}_{(i)},{}^{0}\mathbf{\Gamma}_{0}\right\} = \int_{0}^{T} \mathrm{E}_{(i)}^{2}(t)\cdot dt \tag{7}$$

One can easily see, that χ defines a norm for functions $\mathbf{\Gamma}(t)$ which forms a Hilbert space over this norm.

As we want to lower this norm, we need to alter the $\mathbf{\Theta}_{(i)}$ and $\tau_{(i)}$ input parameters in each step, and leave the other input variables constant. During the steps some boundary conditions (initial and final servo configurations, motion duration, etc.) have to be fulfilled.

We would like to have a monotonous descending series of χ_i elements to reach the optimal input keyframes:

$$(\mathbf{\Theta}_{opt},\boldsymbol{\tau}_{opt}) = \underset{(\mathbf{\Theta},\boldsymbol{\tau})}{\arg\min}\, \chi\left\{\mathbf{\Theta},\mathbf{u},\boldsymbol{\tau},{}^{0}\mathbf{\Gamma}_{0}\right\} \tag{8}$$

In the next section we present a numerical solution for this problem.

3.3 Numerical solution

The goal of the numerical correction algorithm is that the dynamical behavior approximates the reference movement.

Let us define **D** as a subset of input parameter domain $\mathbf{x}_0 := (\mathbf{\Theta}_{Kin},\boldsymbol{\tau}_{Kin})$ with the following properties: **D** shall be the largest connected subset that fulfills all the boundary conditions and contains all the motion parameter combinations where the robot does not fall down and contains the reference movement.

Before applying any numerical optimization method the following presumptions must be taken

$\chi(\mathbf{x})$ shall be continuous and differentiable over **D,** close to the boundaries of **D** the negative gradient $-\nabla\chi$ points inwards,

the optimum $\mathbf{x}_{opt} = (\mathbf{\Theta}_{opt},\boldsymbol{\tau}_{opt})$ lies inside of **D**, there are no other local extremities.

Without the exact mathematical proof, we show that in practical cases these presumptions hold

As long as the robot does not fall down, any infinitesimal change in input parameters effects a proportional infinitesimal change in norm. It is clear that there will be a discontinuity in the norm function where the robot falls.

Nearby the boundaries of **D** the robot motion becomes unstable. The robot body starts to oscillate, therefore the robot segments will have more significant difference from the reference motion, and thus the χ norm grows. Consequently, the negative gradient vector points inwards.

The animator can create a motion, which “seems quite dynamic”; the robot remains standing, and does what he want; thus the motion is in **D** and close to the optimum we search.

It is clear that the final motion will not match the reference motion. It might be impossible to follow it exactly due to ignored dynamics. Therefore, its norm will not reach zero. A local optimum will be where the gradient is zero.

$\chi(\mathbf{x})$ is a strongly non-linear function of its input parameters with narrow potential tunnels, therefore we decided to implement the non-linear conjugated gradient method described in [8].

For this method we needed the gradient of the potential function. We computed it component by component numerically using partial differentials. For the i^{th} component:

$$\nabla\chi(\mathbf{x})_i = \left[\frac{\chi(\mathbf{x}+\Delta\mathbf{x}_i)-\chi(\mathbf{x})}{|\Delta\mathbf{x}_i|}\right], \qquad \Delta\mathbf{x}_i = \pm\varepsilon * \mathbf{e}_i \tag{9}$$

where $\mathbf{e}_i$ is the i^{th} basis vector of the input space. As the **D** is bounded, it can happen that a $\mathbf{x}+\Delta\mathbf{x}_i$ vector points outside of **D**. Then one has to take the inverse of $\Delta\mathbf{x}_i$. In an extreme case if it still points outside of **D**, then we have to renounce the derivative in this direction at the particular step. In this step the dynamical simulation has to be executed (*J*K+K+1*) times.

At initial phase, the algorithm has to perform a line search along the direction of steepest descent. It is an iterative method that should find the minimum along this line. It is a one-dimensional search method. The result is $\mathbf{x}_1$.

After all, the algorithm consists of 5 steps:

1. Compute the gradient in the actual position: $\mathbf{x}_n$
2. Compute β_n according the, Polak–Ribière formula

$$\beta_n = \max\left[\frac{\nabla\chi(\mathbf{x}_n)^T * (\nabla\chi(\mathbf{x}_n)-\nabla\chi(\mathbf{x}_{n-1}))}{\nabla\chi(\mathbf{x}_{n-1})^T\nabla\chi(\mathbf{x}_{n-1})}, 0\right] \tag{10}$$

3. Compute the next conjugated direction

$$\Lambda x_n = \nabla\chi(\mathbf{x}_n) + \beta_n\Lambda\mathbf{x}_{n-1} \tag{11}$$

4. Perform a line search along the last conjugated direction:

$$\min_{\alpha_n} \chi(\mathbf{x}_n + \alpha_n * \Lambda\mathbf{x}_n) \tag{12}$$

5. Next iteration will be then

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_{n*}\mathbf{\Lambda x}_n \tag{13}$$

The algorithm ends if the gradient sinks below a given threshold. Note that the stability reserve of motion is not guaranteed by the algorithm. It is mainly depending on the reference motion.

4 Implementation

At the Department of Control Engineering and Information Technology of Budapest University of Technology and Economics we have a modified version of a KHR-1 humanoid robot (see Fig. 1), the original of which is a commercial product of Kondo Kagaku Co. Ltd., Japan. This experimental biped platform is 34 centimeters tall, has 21 degrees of freedom, and has an onboard control electronic that can interpret only the aforementioned keyframe-based motions.

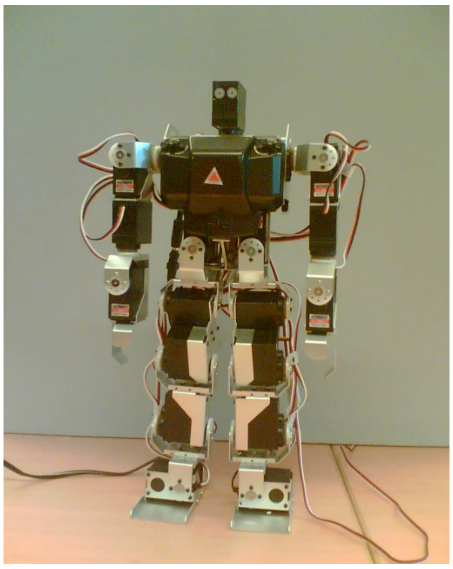

Fig. 1: Our humanoid (Kondo KHR-1)

4.1 Kinematics modeler

We have developed a kinematics-based gait-authoring application for keyframe-controlled robots. This program can be used by an experienced 3D animator to create the keyframes of a desired motion in the virtual world. For this purpose many kinds of interactive tools stay at the user’s disposal (a screenshot of the user interface can be seen in section 5, on Fig. 5).

For each phase of the motion that is currently being edited, one can use the constrained inverse kinematics tools first with the mouse for draft setups, and later fine tune a given group of joints either with the mouse or with the keyboard manually. The length of the individual phases can also be varied, of course.

Assuming our robot is “standing at attention” (we call it as the initial pose), in our example application (see section 5) our goal was to make the robot climb 1-1 steps with both legs over a stair having two steps with length of 5 cm and riser of 3 cm, and then finish movement with the initial pose. We have modeled this locomotion using 9 keyframes (phases), where the startup and the final phase contain exactly the same servo angles. In the meanwhile the startup double support phase (both feet on the floor) transfers to a single support phases (standing on the right foot, swinging the left one and vice versa), and finally we finish in a double support phase again at the top of the stair.

The output of this task is a smooth, harmonic movement in the ideal virtual world (Note, that for the time being we neglect the dynamical behavior of the robot). If we play back this motion on a real robot, it definitely behaves differently. It is predictable, that due to inertial-, contact- and friction forces the real dynamical behavior will issue a pose error at the end, containing two components: our real robot will probably step shorter or longer than 5 cm (position error), and it might deflect from the ideal forward direction (orientation error).

4.2 Using dynamic multibody simulation

In order to overcome the pose error between the realized- and the designed reference motion, we use our presented iterative procedure that reduces this difference. For our method we need a fair dynamic model of the robot.

The Dymola [9] is a multi-engineering modeling and simulation tool, developed by Dynasim AB, Sweden. The multi-engineering capabilities of Dymola presents new and revolutionary solutions for modeling and simulation as it is possible to simulate the dynamic behavior and complex interactions between systems of many engineering fields, such as mechanical, electrical, thermodynamic, hydraulic, pneumatic, thermal and control systems. This means that users of Dymola can build more integrated models and have simulations results that better depict reality. Dymola interprets the declarative object-oriented modeling language Modelica [10], and has interfaces to use additional external user modules written in C or FORTRAN languages.

In Dymola we have built a detailed electro-mechanics model (Fig. 2) of the KHR-1 humanoid:

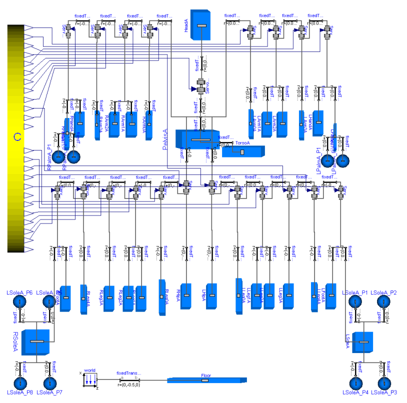

Fig. 2: Humanoid model in Dymola environment

We have developed special building components, which extend the standard models from Modelica.Mechanics.MultiBody library. We had to model contact between objects (and collision response in this manner) and the KRS-784 ICS Digital Servo, which is used in the real KHR-1 robot.

4.2.1 Servo model

The basic actuated revolute joint is encapsulated in a complex servo model (Fig. 3) that contains also the electronic model of the KRS-784. The target angle position control loop is implemented with a simple P controller (the closed loop contains an integrator element, because of the DC motor model). Some parameters of the servo model (e.g.: permanent DC motor’s $V_{nominal}$, $I_{nominal}$, R_a, L_a electromagnetic parameters, rotor inertia, gear ratio, nominal rpm speed) can be found in the data-sheet of the KRS-784, the others and are identified after doing some tests.

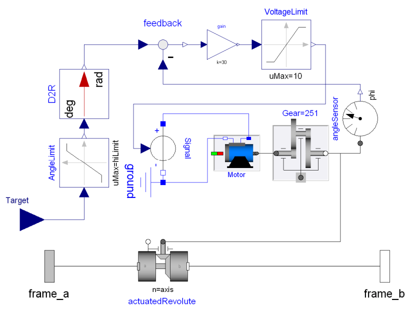

Fig. 3: Our model of a KRS-784 servo

4.2.2 Contact- and collision model

We had to extend the basic Modelica.Mechanics. MultiBody.Parts.Body rigid body model (containing shape, mass, inertia tensor and the Newton/Euler equations of dynamics) with the support of collision handling (which is not included in standard Modelica 2.2.1 libraries at this time – this is the built in Modelica library that comes with Dymola 6.0d).

There are always dynamic constraints between the rigid bodies connected by various joints. Dymola solves the arisen differential algebraic equations (DAEs) and ordinary differential equation sets (ODEs) internally (partially symbolically), where all state variables – including positions and velocities – has to be differentiable. Thus there is no way to use the other popular impulse-based collision response method, which would require sometimes overriding the objects' velocities instantaneously. This is not allowed in Dymola – because this would make the differentiation of velocity vectors not accomplishable. Because of this, we must use a force based method in collision response.

Besides Modelica language, we used partially external C++ implementation with the popular SOLID interference detection library [11], which can be used to retrieve contact points between pairs of objects (it uses the GJK algorithm [12]), but it does not calculate the response, by default. We made a spring and damper material model, and calculate the contact force in normal direction (along the vector defined by two contact points) the following way:

$$|\boldsymbol{F}_{NORMAL}| = \begin{cases} 0, & \boldsymbol{p} < 0 \\ \left[1 + \dfrac{1-\varepsilon}{\varepsilon \cdot v_{COLL}} \cdot \dot{\boldsymbol{p}}_{\perp}\right] \cdot \boldsymbol{S} \cdot \boldsymbol{p}, & \boldsymbol{p} \geq 0 \end{cases} \tag{14}$$

The scalar 'p' means the penetration depth [m]. If we project the relative velocity of the contact points to normal direction vector, we get the signed $\dot{p}_{\perp}$ [m/s] component of penetration velocity, the value of which is stored in 'v_{COLL}' at the moment of first contact. The spring coefficient is 'S' [N/m] and the restitution factor is ε in the previous formula.

When the relative velocity of two interpenetrating bodies have nonzero tangential component (v_t), it is very important that we calculate friction forces using a friction model. Without this effort our virtual robot won't be able to make any translational movement at all. These forces are parallel to the plane, the normal of which is the vector between the two contact points. The friction model is the following:

$$|\boldsymbol{F}_{FRICTION}| = \begin{cases} \mu_{kin} \cdot |\boldsymbol{F}_{NORMAL}|, & |\boldsymbol{v}_t| > v_{st} \\ \mu_{stat} \cdot \dfrac{|\boldsymbol{v}_t|}{v_{st}} \cdot |\boldsymbol{F}_{NORMAL}|, & |\boldsymbol{v}_t| < v_{st} \end{cases} \tag{15}$$

We have two friction coefficients for the static and for the kinetic cases. The constant speed value v_{st} represents the limit, which influences whether objects are considered as sliding or remain resting. The result of these two forces will act in opposite directions on both objects in each colliding pair.

4.2.3 Implementing our iterative enhancement method for the realized motion

We implemented the iterative algorithm also in Dymola environment. Fig. 4 explains the block scheme of the implementation, with the three main software components:

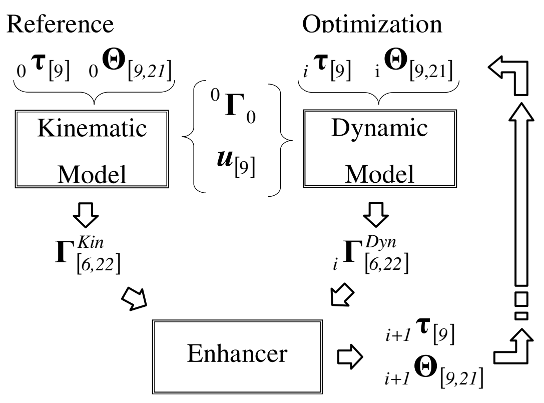
Reference
Optimization
$_0\boldsymbol{\tau}_{[9]}$ $_0\boldsymbol{\Theta}_{[9,21]}$
$_i\boldsymbol{\tau}_{[9]}$ $_i\boldsymbol{\Theta}_{[9,21]}$
Kinematic Model
$^0\boldsymbol{\Gamma}_0$
$\boldsymbol{u}_{[9]}$
Dynamic Model
$\boldsymbol{\Gamma}^{Kin}_{[6,22]}$
$_i\boldsymbol{\Gamma}^{Dyn}_{[6,22]}$
Enhancer
$_{i+1}\boldsymbol{\tau}_{[9]}$
$_{i+1}\boldsymbol{\Theta}_{[9,21]}$

Fig. 4: The three main modules of the implemented algorithm

The "Kinematic Model" module serves the reference pose-time functions of all robot segments using linear keyframe interpolation. The "Enhancer" module can query the pose of a given segment at any time instant between 0 and T (the simulation length).

In all iterations the "Dynamic Model" module calculates actual pose functions for the segments, using the internally constructed dynamic model. The Enhancer analyses the difference of these outputs and calculates the new input keyframes (joint angles and interval-lengths) according to the method presented in sections 3.2 and 3.3. The output of the Enhancer is fed back to the Dynamic Model, thus forming a closed loop of iterative motion enhancement procedure.

5 Application and results

We tested our locomotion enhancement algorithm on the KHR-1 humanoid doing a short stair climbing sequence (see Fig. 5 for a keyframe in our modeler application).

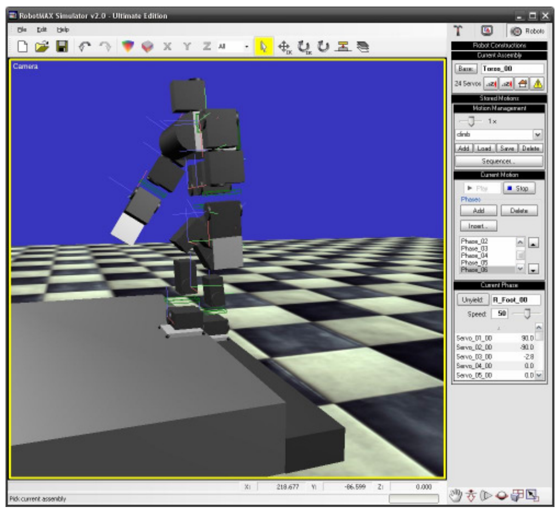

Fig. 5: A screenshot of the modeler application

Assuming we start with parallel feet at the base level of the stair, our goal was to make the robot climb 1-1 steps with both legs. Finally we wanted the feet to finish side-by-side, again. We built a quite steep stair with step of 5 cm and riser of 3 cm, thus the demanded locomotion takes 10 cm forward while the robot is ascending 6 cm.

In our kinematic modeler we approximated this motion with 9 keyframes, resulting in a total length of 1.5 seconds. After playing the original motion on the robot, it went 1.3 cm askew and turned about 7 degrees right.

For the enhancement procedure the **W** diagonal weight matrix was set to identity in position and 0.1 in orientation elements, so that angular errors were punished equally to the arc length of a 10 cm long section (average height of center of gravity). The simulation length (T) was set 50% longer than the total motion time, in order to incorporate the final robot body oscillation at the end, which should also have been decayed.

The iterative motion correction algorithm runs very long. Although a single run of the dynamic simulation for the humanoid takes 0.5 s in Dymola on our PC, computation of the gradient requires 199 simulation runs ($1 + K*J+K$ times, where K=9, J=21) – which means about 1.5 minutes. In addition the line search method requires 40 additional iterations, thus a single enhancement cycle requires ca. 2 minutes (one step along the conjugate gradient). The dimensionality of this highly non-linear system needed 22 x 9 = 198 cycles, so finally the total simulation time was approximately 6.5 hours (because it has $O(N^2)$ time complexity).

Fig. 6 shows a snapshot (taken at the same time instance as Fig. 5) in Dymola environment, visualizing the contact forces at the feet, too:

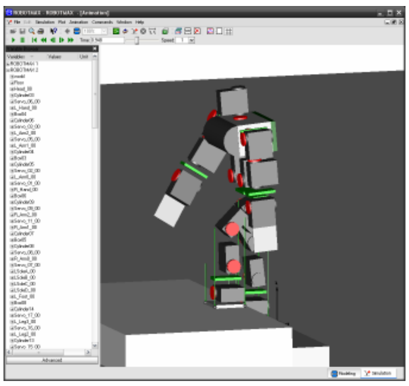

Fig. 6: A screenshot taken in Dymola during simulation of the final motion

In return, the final pose error of the dynamical motion became minimal, thus we proved that this algorithm can enhance the realized motion well.

6 Acknowledgement

We would like to acknowledge the European Commission and the Fraunhofer Institute IFF for hosting the Marie Curie Early Stage Researcher Fellowship Program, furthermore the Hungarian Scientific Research Fund grant No.: T-042634OTKA and the EuroPR Computers Ltd for their kind support in practical exploitation of the research results.

7 References

[1] T. Juhász, T. Urbancsek. *Beyond the Limits of Kinematics in Planning Keyframed Biped Locomotion*, Periodica Polytechnica Electrical Engineering, Budapest, 2007, (accepted)

[2] R. Bohlin and L. Kavraki. *Path planning using Lazy PRM.* In Proc. IEEE Int. Conf.Robot. & Autom. (ICRA), April 2000. version 5. The Mathworks Inc., Natick, 1998.

[3] S.M. LaValle and J.J Kuffner. *Randomized kinodynamic planning.* In Proc. IEEE Int Conf. Robot. & Autom. (ICRA), May 1999.

[4] J.J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. *Footstep planning among obstacles for biped robots.* In Proc. IEEE/RSJ Int. Conf. Intell. Robot. & Sys. (IROS), October 2001.

[5] J.J. Kuffner, S.Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. *Dynamically-stable motion planning for humanoid robots.* Autonomous Robots (special issue on Humanoid Robotics), 12(1):105–118, 2002.

[6] M. Girard. *Interactive design of computer-animated legged animal motion.* IEEE Computer Graphics & Applications, 7(6):39–51, June 1987.

[7] M. Vukobratovic, B. Borovac, D. Surla, and D. Stokic. *Biped Locomotion: Dynamics, Stability, Control, and Applications.* Springer-Verlag, Berlin, 1990.

[8] Wikipedia *Nonlinear conjugate gradient method* http://en.wikipedia.org/wiki/Nonlinear_conjugate_gradient

[9] Dynasim AB: *Dymola – Dynamics Modeling Laboratory:* http://www.dynasim.com

[10]Modelica: http://www.modelica.org

[11]SOLID - *Software Library for Interference Detection:* http://www.dtecta.com

[12]E. G. Gilbert, D.W. Johnson, and S. S. Keerthi. *A fast procedure for computing the distance between complex objects in three-dimensional space.* IEEE Journal of Robotics and Automation, 4(2):193–203, 1988.

[13]N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. *Choosing good distance metrics and local planners for probabilistic roadmap methods*. IEEE Trans. Robot. & Autom., 16(4):442–447, August 2000.