Preprint of the paper which appeared in: Proc. IEEE Intl. Conf. on Mechatronics and Robotics (MechRob), Sept. 13-15, Aachen, Germany, pp. 972-977, Sascha Eysoldt Verlag, 2004.

Object-oriented Dynamics Modeling for Legged Robot Trajectory Optimization and Control

Robert Höpler *, Maximilian Stelzer, and Oskar von Stryk
Simulation and Systems Optimization Group
Technische Universität Darmstadt
D-64289 Darmstadt, Germany
[hoepler|stelzer|stryk]@sim.tu-darmstadt.de
http://www.sim.informatik.tu-darmstadt.de

***Abstract*— This paper discusses the development and application of object-oriented modeling and implementation techniques to achieve a representation of the mechanical model amenable to the various requirements by legged robot applications. This leads to a uniform, modular, and flexible code generation while reaping the performance of efficient domain-specific articulated body algorithms. Trajectory optimization problems of bipedal and quadrupedal robots are investigated as applications. It is shown how a high-level specification of multibody dynamics models using component libraries serves as a basis for generation of a number of modules forming an "overall" computational dynamics model. The selected examples illustrate how one copes with the emerging complexity by integrating various modules for, i. e., equations of motion, non-linear boundary conditions, symmetry and transition conditions, for multiple phases during motion. Variations of the mechanical structure, e. g., contact conditions can be treated either by multiple models or by reconfiguration of one model. Numerical results are presented for time- and energy-optimal trajectories of full three-dimensional models of a humanoid robot and a Sony four-legged robot.**

I. Introduction

The development of flexible, dynamic and autonomous (i. e. guided by external and internal sensor data) motions for quadrupedal and humanoid robots is still a challenging and mainly unsolved task. The dynamics of legged robots is characterized by a free-floating, tree-like multibody systems structure with a high number of degrees of freedom consisting of many links and many actuated joints. Due to the frequent changes of the contact situation during legged locomotion frequent changes in the dynamic model occur. This special structure can usually not be utilized for an efficient implementation when using general purpose multibody systems (MBS) formalisms and tools. To cope with the high complexity of the nonlinear dynamics of legged robots model-based methods for real-time actuator control, for trajectory optimization, and for controller design specifically tailored to the *one* scenario of legged robots must be developed. For rapid and virtual prototyping a most desireable goal is not only to apply the same abstract dynamic model representations and mathematical models but also as many parts as possible of the same *code* for off- and on-line evaluations of legged robot dynamics during *all* stages of design, development, implementation and operation of a legged robot. To facilitate the investigation of new concepts of nonlinear model-based optimization and control methods also the sensitivities of the legged robot dynamics model with respect to its state variables and parameters are needed. The formalisms and tools applied at the same time have to cope with (i) the complex underlying mechanical model, characterized by many degrees of freedom, actuator dynamics, and interaction of the robot with its physical environment including time-varying contacts and collisions, (ii) the wide range of required numerical schemes, including kinematics, dynamics, sensitivity information, etc., (iii) efficient code generation which is particularly vital for on-line computations, and must rely on the power of dedicated (recursive) algorithms, and (iv) interaction of the computational model with the system it is running on, e. g., communication with sensors and actuators.

*Formerly with the Institute for Robotics and Mechatronics, German Aerospace Center (DLR)

II. Dynamics modeling of legged robots

Legged robots are described as free-flying, fully three-dimensional rigid MBS experiencing contact forces to allow for the modeling of walking or running gaits including phases with all legs in a flight phase and for the application of various ground contact models, though actually feet-ground interaction is modelled by completely inelastic collision.

A. Forward dynamics model

The joint space equations of motion of a rigid multibody system in the presence of ground contact, i.e. tip contact forces and holonomic tip constraints, are

$$\ddot{\mathbf{q}} = \mathcal{M}^{-1}(\mathbf{q})\left(B\mathbf{u} - \mathcal{C}(\mathbf{q},\dot{\mathbf{q}}) - \mathcal{G}(\mathbf{q}) + J_c^T(\mathbf{q})\,\mathbf{f}_c\right) \quad (1)$$

$$0 = \mathbf{g}_c(\mathbf{q}), \quad (2)$$

where $\mathbf{q}, \dot{\mathbf{q}} \in \mathbb{R}^{n_d}$ are the column matrices of joint position and velocity variables, $\mathcal{M}$ is the positive-definite joint space mass matrix, $\mathcal{C}$ and $\mathcal{G}$ are the vectors of gyroscopic and gravitational forces, the vector $\mathbf{u} \in \mathbb{R}^m$ is the vector of actively controlled joints, which is mapped with the constant matrix B. The n_c holonomic ground contact constraints $\mathbf{g}_c \in \mathbb{R}^{n_c}$ result in a constraint Jacobian $J_c = \frac{\partial \mathbf{g}_c}{\partial \mathbf{q}} \in \mathbb{R}^{n_c \times n_d}$ and $\mathbf{f}_c \in \mathbb{R}^{n_c}$ is the vector of ground constraint forces.

B. Reduced dynamics

The numerical difficulties associated with the system of differential algebraic equations (DAEs) of high index, resulting from the general modeling approach of multibody dynamics and algebraic equations for contact, can be avoided. This is done by a reduced dynamics method [1], [2], treating explicitly only the independent states $\mathbf{q}_I$, which are global orientation and position and states related to legs in contact with the ground, and using inverse kinematics to determine the dependent states $\mathbf{q}_D$ of the other legs:

$$\begin{aligned} \mathbf{q}_I &:= \text{global orientation, position; swing leg(s) states} \\ \mathbf{q}_D &:= \text{contact leg(s) states} \end{aligned}$$

$\mathbf{q}_I$ may be computed from all states $\mathbf{q}$ using a constant mapping Z, i.e. $\mathbf{q}_I = Z\mathbf{q}$. The solution of the resulting reduced system of ordinary differential equations (ODEs)

$$\ddot{\mathbf{q}}_I = Z\mathcal{M}\left(\tilde{\mathbf{q}}\right)^{-1}\left(B\mathbf{u} - \mathcal{C}\left(\tilde{\mathbf{q}}, \dot{\tilde{\mathbf{q}}}\right) - \mathcal{G}\left(\tilde{\mathbf{q}}\right) + J_c^T(\mathbf{q})\,\mathbf{f}_c\right),$$

where $\tilde{\mathbf{q}}$ consists of the independent states and of the dependent states determined from inverse kinematics on position and velocity level, then may be proven to be the solution of the initial system of differential algebraic equations [1].

C. Sensitivity computations

Sensitivity computations are essential in problems involving optimization, non-linear analysis, and linearization. Linearized forward and inverse dynamics models for robots are useful in motion planning and control applications [3]. The objectives in trajectory optimization of legged robots to apply sensitivities of the forward dynamics model are (i) more robust and faster convergence in gradient based methods and (ii) more reliable approximation of the Hessian from the exact analytical first derivatives. In contrast to numerical differentiation this technique is robust, avoids errors, and is scalable to large-dimensional systems such as full three dimensional humanoid models. For trajectory optimization the sensitivity of the state space forward dynamics model (1) w. r. t. control and state variables is required:

$$\delta\ddot{\mathbf{q}} = \nabla_{\mathbf{u}}\ddot{\mathbf{q}}\,\delta\mathbf{u} + \nabla_{\mathbf{q}}\ddot{\mathbf{q}}\,\delta\mathbf{q} + \nabla_{\dot{\mathbf{q}}}\ddot{\mathbf{q}}\,\delta\dot{\mathbf{q}} \tag{3}$$

D. Dynamics algorithms

The numerical evaluation of forward dynamics is performed in terms of $\mathcal{O}(N)$ Articulated Body Algorithm (ABA) [4]. This is an efficient and numerically stable algorithm for moderately constrained tree-structured systems with many degrees of freedom. The contact constraint forces and reduced dynamics are calculated using methods described in [1] applying the operational space inertia matrix [5]. This inertia matrix serves as a basis for the solution of collision dynamics, too. It is used to calculate the amount of impulse when tips of the robot collide with the ground. The method to distribute the impulse over the MBS is described in [6]. The algorithms applied to calculate the linearized forward dynamics model are based on differentation of a recursive symbolic forward model [3]. The numerical evaluation of $\delta\ddot{\mathbf{q}}$ is of order $\mathcal{O}(N)$. All algorithms are of recursive nature and well-suited for efficient object-oriented implementation as shown below.

III. Trajectory optimization

A. Optimal control problem

Finding stable gaits for walking robots with four or two legs is still challenging due to the high complexity and redundancy of the underlying mechanical structure. Heuristic methods, e.g., inverted pendulum methods, usually do not consider the full dynamics of the robot as on-line computing of walking trajectories is computationally too expensive. Stating the problem of finding periodic and statically or dynamically stable gaits for legged robots off-line avoids the problem of high on-line computational costs and allows for application of full three-dimensional dynamics models in the computation.

Different gait patterns (such as walk, trot, rack, canter and rotary or transverse gallop for four-legged robots or walk and run for biped robots) differ in the duty factor of each foot, i.e. the fraction of a total stride cycle during which the foot is in contact with ground, and relative phases of feet, i.e. the order and time displacement of feet reaching ground [7].

The optimal control problem is stated as follows [2]:

$$\begin{aligned} \min \mathcal{J}[\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}, t_f] \quad \text{s. t.} \quad & \text{minimize the merit} \\ & \text{function } \mathcal{J} \text{ subject to} \\ \mathcal{M}\ddot{\mathbf{q}} = B\mathbf{u} - \mathcal{C} - \mathcal{G} + J_c^T\mathbf{f}_c \quad & \text{system of MBS ODEs} \\ \mathbf{g}_c\left(\mathbf{q}\right) = 0 \quad & \text{contact algebraic conditions} \\ \mathbf{b}\left(\mathbf{q}_0, \mathbf{q}_f, t_0, t_f\right) = 0 \quad & \text{boundary conditions} \\ \mathbf{n}(\mathbf{q}, \mathbf{u}) \geq 0 \quad & \text{nonlinear inequality constr.,} \\ \mathbf{q}_{\min} \leq \mathbf{q} \leq \mathbf{q}_{\max}, \quad & \text{box constraints on state} \\ \mathbf{u}_{\min} \leq \mathbf{u} \leq \mathbf{u}_{\max} \quad & \text{and control variables.} \end{aligned}$$

The original DAE system (1) and (2) can be replaced for the numerical solution by the reduced dynamics equations of II-B. Merit functions $\mathcal{J}$ may, e.g., be time t_f, energy or a weighted sum of both [2]. Boundary conditions at initial, intermediate and/or final time of a gait cycle contain conditions for

- symmetry resp. anti-symmetry of states,
- foot placement, i.e. conditions that force the feet to be placed on desired positions (which may depend on parameters and therefore may also be subject to the optimization),
- contact forces at the end of a stance phase, that allow the foot to lift off

(see [2] for more details). Nonlinear inequality constraints are:

- Hips of legs in contact with the ground must stay within a maximum radius of the leg, so that the inverse kinematics solution required for reduced dynamics has a well-defined solution.
- The swing feet are kept above the ground according to a given contour, for example a proper sine curve. This property increases stability by avoiding contact with the ground resulting from non-modeled deflexions of bodies and joints, which could lead to stumbling of the robot.
- Slipping is avoided by limiting the horizontal contact forces relative to the vertical contact forces.
- Vertical contact forces must be positive, i.e. the robot may only push to ground but may not pull from ground.

- Further constraints to be considered are detailed motor characteristics. By now the box constraints for minimal and maximal values of angular velocities and torques only give a rough estimate of the real actuator data.

Stability may be enforced explicitly by inserting any common criteria into the optimal control problem. A more detailed discussion of the constraints can be found in [8] for a humanoid model and in [9] for a four-legged robot.

B. Optimization method

For solving the optimal control problem numerically, the method DIRCOL [10] is used. The states and controls are approximated by piecewise cubic resp. piecewise linear polynomials on a discrete and successively refinable time grid. The optimal control problem is thereby transcribed into a nonlinear program with the coefficients of the polynomials as variables, which may be solved by a – due to the special structure of the variables – sparse sequential quadratic programming method [11]. For more details we refer to [10], [12].

IV. Object-Oriented Modeling Architecture

Practical realization of models for non-linear dynamics computations of legged robots must consider (i) the complexity of the mechanical structure, (ii) a great variety of components and actuation methods, (iii) demanding environmental conditions, (iv) types, efficiency, and interaction of available kinematics and dynamics computer algorithms suitable for such systems, and (iv) application scenarios reaching from off-line trajectory optimizations to real-time closed-loop control including integration and communication to external soft- and hardware with tight timing constraints. Based on a detailed analysis this section proposes an object-oriented architecture satisfying the requirements from the legged robot trajectory optimization application leading to (i) modular, (ii) efficient, (iii) consistent, and (iv) reconfigurable characteristics.

A. Requirements analysis

The requirements for a software system can be captured by a system model which describes what to realize and how. This model forms an abstraction in two ways [13], [14]. First, it abstracts from real world details which are not relevant for the system. Second, it also abstracts from the implementation details and hence precedes an actual implementation in a programming language.

The ultimate goal of the architecture is to implement multibody kinematics and dynamics algorithms efficiently. The primary task is to investigate which type of computations will be required. For brevity we restrict here to the trajectory optimization problem discussed in III. The trajectory optimization algorithm requires the set of equations of motion of the robotic system over one *complete* gait cycle, i. e., including time-varying support phases reflected by a changing, phase dependent dynamics model. The first part required is a forward dynamics model which can efficiently be implemented by dedicated composite rigid body and articulated body algorithms [15], [4]. Depending on the current support phase or contact state a contact dynamics model is applied to consider the additional contact constraints and to reduce to state space form [6]. The reduced dynamics requires forward- and inverse kinematics on position-, velocity-level. The treatment of a complete gait cycle as a multi-phase dynamics problem requires additional collision computations to model the discontinuous state transitions between various support phases. Depending on the applied optimization method derivatives of the equations of motion w. r. t. state variables and design parameters might be required. When optimization is subjected to additional stability criteria new types of computations must be introduced to determine, e. g., the center of mass or the zero moment point of the legged robot depending on the current state $\mathbf{q}$.

All computations reflect certain physical aspects of the *same* legged machine in certain states leading to a strong coupling between all parts. Consequently there must exist a common representation of the mechanical model and the problem setup. An abstract high-level description could serve as a basis for all types of required computations. There is the need to support many different types of legged machines, such as humanoids, four-legged walking machines, and a large variety of components forming the mechanical structure, different actuators, and contact models.

Computational efficiency is of high importance in trajectory optimization and most control applications of realistic legged robots. The complexity of the governing dynamics equations for full three dimensions, the large number of degrees of freedom, and the possibly large number of dynamics evaluations during iterations of trajectory optimization methods makes this a challenging problem. As a consequence, the most efficient and robust MBS algorithms must be chosen which often are problem specific solutions. The most prominent example is the analytical solution of the inverse kinematics problem, which gives fast and reliable solutions when compared to a general purpose approach. To integrate in one formalism general purpose and problem specific algorithms, which have been generated either automatically or manually coded, a flexible software architecture is indispensible.

B. Design of the architecture

We propose to reconcile these concurrent requirements by a carefully designed object-oriented operational architecture. The goal is to enable easy implementation and integration of dynamics computations but not to create a general purpose multibody program. This comprises a

- high-level and component-oriented description of the robot model and problem setup, called *specification model*
- objects serving as domain-specific code generators mapping the descriptions creating
- objects serving as encapsulated algorithms with clear interfaces and semantics.

1) Specification model: Because legged systems are quite complex mechanisms it is beneficial to use a high-level model

description being modular and hierarchical. Components belonging to the multibody system domain, such as links, bodies, joints, drives, are classified in an abstract way as *MBS entities.* In object-oriented terms these are descendants of the class *MBSEntSpec*. The domain component library contains a carefully selected, finite set of components representing concrete mechanical parts, such as drive-trains, or new mathematical models such as for contact constraints.

The topology of the mechanical model is formed by defining relations, called *MBS connections,* between certain ports being part of each MBS entity. These interaction ports belong to the class *MBSPortSpec*, the MBS connection is represented by class *MBSConnectionSpec*.

The set of components and relations between the ports forms an a-cyclic graph where the edges are the connections and the vertices are the MBS entities. This is the specification of the mechanical model, which is represented by class *MBSModelSpec*. There is an important semantic issue to note. Algorithms often are not valid for the complete model, but only for parts, e. g., inverse kinematics or the Jacobian for one single leg. In order to keep all computations consistent the model specification aggregates *references* to MBS entities and MBS connections ensuring that all algorithms work consistently on the same data and topology. That is crucial in walking robot scenarios where changing structural conditions require reconfiguration of the models or after a calibration cycle.

The desired activity also is considered part of the problem description, i. e., the type of computation, is determined by *solver specification*s. These are objects of class *SolverSpec*. These are parametrized by (i) the expected behaviour, (ii) the type of algorithm applied, and (iii) further tags to denote for instance the coordinate representation used in the computations. The SolverSpec shown in the last line in Fig. 2 represents an articulated forward dynamics algorithm using body-fixed coordinates, e. g., see [16].

The specification model of the legged robot, however, does not provide any executable code. The purpose is exclusively to decouple dynamics model specification from its implementation and to form an object-oriented basis for other abstract representations such as textual or graphical representations. This additional level of abstraction is amenable to formal analysis, e. g., for verification or optimization of the specification w. r. t. to certain criteria, or when making the state of the system persistent.

2) Operational model: This section introduces two additional types of objects, the *Builder* and the *Solver*, reaping the benefits from the abstract model specification. The main purpose of a builder is to automatically transform, to *map*, the specification model to one or more products, the solvers. Solvers discussed in this work are executable units of code, instances performing the desired computations. This behaviour of translating a high-level information into low-level implementation makes a builder object a domain-specific *application generator* [17]. The main difference is that a builder is able to immediately create or reconfigure executable solver instances and does not need to invoke a time- and resource-consuming compile-to-code step. The classes representing the specification and the operational model and their relations are shown in Fig. 1.

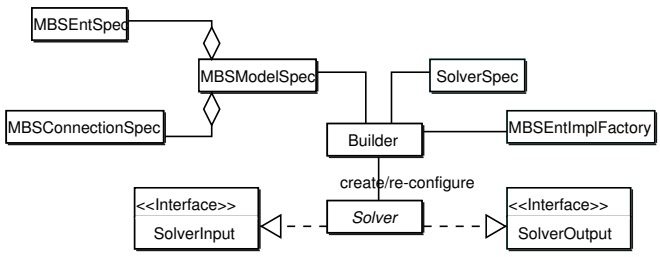

Fig. 1. UML class diagram showing the relations between the main classes of the robot modeling framework. The central parts are the Builder and Solver objects. The former converts a given MBS model specification into an executable solver instance able to perform the desired dynamics calculations.

The transformation process performed by a builder without compile-to-code step is as follows. A builder first checks if the specification and context information meets its requirements to create the output solver. This includes a check for correctness of the model description. In the next step a builder may create a intermediate specifications, e. g., to improve model properties largely to enhance run-time characteristics. Finally it assembles executable objects containing code solving parts or the complete set of the required mathematical equations. These objects of class *MBSEntImpl* represent the multibody domain knowledge of the robotics engineer, coded and precompiled, implementing a well-defined interface. Objects of class *MBSEntImplFactory* supply the builder with MBSEntImpl objects in the demanded form. This can be viewed as the data base for pre-coded component equations and in most cases it is realized using a factory pattern [18].

From a specification perspective [19] a solver represents an algorithm interface which is implemented by one single MBSEntImpl or a complete dataflow network of interconnected components. In this case a solver acts like the Multi-Graph-Kernel known in the Model-Integrated-Computing approach [20], [21], [22] conducting computations by controlling the dataflow in a dataflow process network. This approach of distributing the computations has two disctinct advantages: It enables either possible reuse of equations and code at compile-time or reuse of numerical results during run-time through solver coupling.

V. Results

A. Architecture for gait trajectory optimization

Application to trajectory optimization leads to a package structure depicted in Fig. 3. The DIRCOL optimizer interface comprises three main blocks depending on the system under investigation, the differential equation, the linear- and non-linear boundary conditions, and behaviour when switching between various leg support phases. In our realization each is satisfied by a set of coupled solver objects.

The model parts of the first package 'Differential equations' are implemented by one solver containing an $\mathcal{O}(N)$ articulated body algorithm for tree-structured systems [4], a collection of hand-tailored inverse kinematics solvers for each single

```
// Problem specification block
RevoluteJoint joint1;
RigidLink link1;
MBSConnectionSpec connection1(joint1.port2,link1.port1);
...
MBSModelSpec leg1,leg2,torso,humanoid;
leg1.add(joint1);
leg1.add(link1);
leg1.add(connection1);
...
humanoid.add(leg1,leg2,torso);
// Builder block
SolverSpec<FwDyn,ABA,BF> DynSpec;
Builder<DynSpec,...> builder;
Solver<DynSpec,...> dynamics=builder.create(humanoid);
...
```

Fig. 2. Example C++ pseudo-code exemplifying problem specification and solver creation. The first block sketches the specification of components and topology of a humanoid and its constituents. The second block shows how a builder creates a solver doing forward dynamics (FwDyn), using an articulated body algorithm (ABA) and body-fixed coordinates (BF).

leg and Jacobian or operational space inertia solvers for the velocity inverse kinematics. The contact state is represented by a solver depending on the time-varying contact model specification. The second package 'Constraints' contains solvers wich compute the center of gravity of the complete mechanical system, and one solver to detect foot collisions with ground and between legs. The task of the third package 'Phase switching' is to react on requests from DIRCOL to change the support phase by invoking a collision solver. The contact state is changed and the system of solvers reconfigured accordingly to keep all components consistent. DIRCOL is able to process user-defined gradient information. In order to enhance convergence properties an optional set of solvers provides exact sensitivities of the differential equations. The actual implementation supports sensitivities for robots with fixed base.

Creation of this dynamics model requires three steps: First, creating objects representing the mechanical components applied for model specification are rigid links, revolute and six-dof joints, contact points, and drives. Second, creating descriptions comprising single legs, needed for inverse kinematics, the complete free-flying legged mechanism, needed for the reduced dynamics algorithm, and the complete mechanism including contact and collision interaction with the environment, for reduced dynamics and phase switching. Third, all solvers mentioned are instantiated and reconfigured by a collection of dedicated builder objects from these unique high-level specifications using solver specifications to denote the desired computations.

B. Trajectory optimization experiments

This section shortly reviews results of the application of the presented approach for legged robots. Walking trajectories involving full three-dimensional models have been calculated both for a MBS model of a four-legged robot [9] and a two legged humanoid robot [8] (cf. Fig. 4). In [8] and [9] originally SOAFOR has been used. This is a set of Fortran subroutines implementing articulated body algorithm and reduced dynamics for legged robots applying spatial operator algebra and has evolved from [1] but does not provide sensitivities.

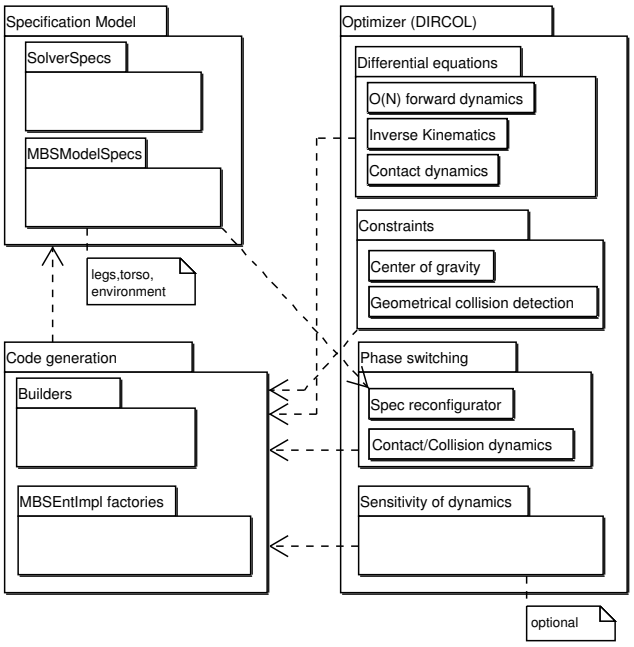

Fig. 3. UML package diagram of gait trajectory optimization within DIRCOL using Spec-, Builder-, and Solver objects.

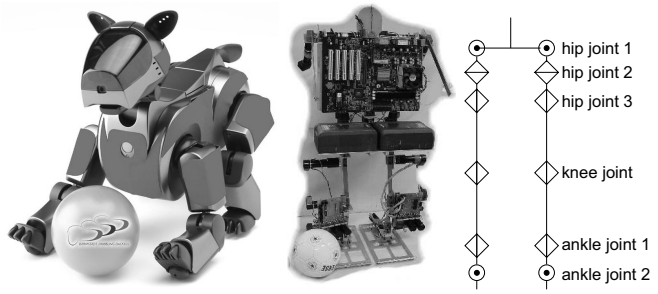

Fig. 4. Left: Four legged Sony Aibo robot. Center: prototype of a humanoid robot. Right: Kinematic structure of humanoid model. For both robots optimal walking trajectories have been calculated.

Here, we demonstrate that these solutions also can be obtained with almost the same efficiency using the much more flexible object-oriented modeling approch presented here.

For Sony's four-legged robot model ERS-210A with legs of three degrees of freedom each, dynamics model data has been provided by the manufacturer. First optimal trajectories lead to successive refinements of the model: modified box constraints of applied torques and angular velocities of the joints lead to better fit the measured robot trajectories (5) and slipping of the robot's feet on the ground was avoided by adding constraints on the horizontal contact forces [9]. Different symmetric gaits like trot and walk have been calculated.

The humanoid robot model, one torso and two 6 dof legs, of a real prototype developed in a collaboration including our group [8] was applied for optimization of a two-phase half-stride consisting of a single limb support phase and a double limb support phase. Static stability explicitly is enforced by nonlinear inequality constraints. One dynamics evaluation for this system according to V-A requires about 1.6 ms on a 2 GHz Pentium including exact sensitivities during smooth phases of the trajectory. As current humanoid control cycle times are around 1 ms, the approach presented in this paper now

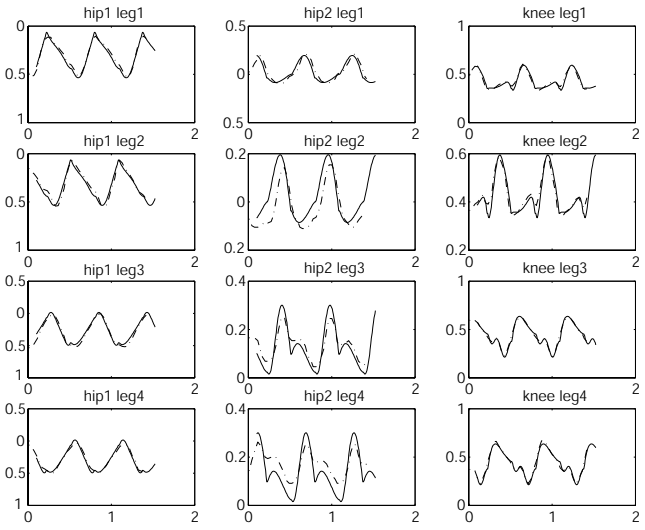

Fig. 5. The experimentally measured joint angle trajectories (dotted lines; joint angle [rad] versus time [s]) for the first hip joints and the knee joints match the computed reference trajectories (solid lines) quite well after considering improved estimates for maximum torque and velocity constraints. For the second hip joints, the constraints have not yet been adapted resulting in the depicted difference. The joint trajectories are shown for about two and half strides of the trot gait.

enables the investigation of new nonlinear dynamics model-based control methods for humanoid robots by modular and flexible generation and re-use of code.

The optimized gait pattern for the four-legged robot is a *trait*, which is a symmetric gait defined by diagonal legs moving synchronously. When exploiting this symmetry only one phase has to be considered, because the contact situation does not change for the trot during one half-stride. In both cases the obtained gaits obey energy-optimality.

VI. Conclusion

This paper has presented a new object-oriented architecture to provide multibody dynamics model computations which are indispensible for robot control applications. The main application considered for demonstration is trajectory optimization of legged robots. The approach consists of a carefully designed class hierarchy which separates the specification of the mechanical model and the desired calculations, referred to as specification model, from the implementation part. The latter consists of algorithm-specific code-generators (Builder) creating executable instances (Solver) finally performing the desired multibody computations. The two main advantages for the usage of Builder/Solver components are on the one hand the application of dedicated efficient problem-specific algorithms and closed-form solutions to reap maximum efficiency and not to rely on one general purpose formalism and to allow for flexible algorithm interchange. This results in an efficient and light-weight code without a compile-to-code step required. On the other hand the specification model permits more general topologies, components, and algorithms, indispensible for further research which will be directed to legged robot models of increased complexity, including arm dynamics and advanced actuator concepts using artificial muscles.

Acknowledgement

The first author has been supported by the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), for research in object-oriented modeling of manipulator kinematics and dynamics during the last years.

References

[1] M. W. Hardt, "Multibody dynamics algorithms, numerical optimal control, with detailed studies in the control of jet engine compressors and biped walking," Ph.D. dissertation, University of California, San Diego, 1999.

[2] M. W. Hardt and O. von Stryk, "Dynamic modeling in the simulation, optimization, and control of bipedal and quadrupedal robots," *Z. Angew. Math. Mech.*, vol. 83, no. 10, pp. 648–662, 2003.

[3] A. Jain and G. Rodriguez, "Linearization of manipulator dynamics using spatial operators," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 1, pp. 239–248, 1993.

[4] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *The Int. J. Robotics Res.*, vol. 2, no. 1, pp. 13–30, 1983.

[5] O. Khatib, "Dynamic control of manipulators in operational space," in *Proc. 6th CISM-IFToMM Congress on Theory of Machines and Mechanisms*, New Delhi, India, Dec. 1983.

[6] D. Agahi and K. Kreutz-Delgado, "A star topology dynamic model for efficient simulation of multilimbed robotic systems," in *Proc. IEEE Conf. on Robotics and Automation*, 1994, pp. 352–357.

[7] R. M. Alexander, "The gaits of bipedal and quadrupedal animals," *The Int. J. Robotics Res.*, vol. 3, no. 2, pp. 49–59, 1984.

[8] M. Buss, M. Hardt, J. Kiener, M. Sobotka, M. Stelzer, O. von Stryk, and D. Wollherr, "Towards an autonomous, humanoid, and dynamically walking robot: Modeling, optimal trajectory planning, hardware architecture, and experiments," in *Proc. 3rd IEEE/RAS Intern. Conference on Humanoid Robots, Karlsruhe and München, Germany, Oct. 1-3*, 2003.

[9] M. Stelzer, M. Hardt, and O. von Stryk, "Efficient dynamic modeling, numerical optimal control and experimental results for various gaits of a quadruped robot," in *CLAWAR 2003: Intern. Conf. on Climbing and Walking Robots, Catania, Italy, Sept. 17-19*, 2003, pp. 601–608.

[10] O. von Stryk, *User's Guide for DIRCOL – a Direct Collocation Method for the Numerical Solution of Optimal Control Problems, Version 2.1*, rev. ed., Technische Universität Darmstadt, 2002.

[11] P. Gill, W. Murray, and M. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Journal on Optimization*, vol. 12, pp. 979–1006, 2002.

[12] M. Hardt and O. von Stryk, "Towards optimal hybrid control solutions for gait optimization patterns of a quadruped," in *Proc. 3rd International Conference on Climbing and Walking Robots, CLAWAR 2000*. Madrid: Bury St. Edmunds and London, UK: Professional Engineering Publishing, 2000, pp. 385–392.

[13] D. Hatley and I. Pirbhai, *Strategies for Real-Time System Specification*. New York: Dorset House Publishing Company, 1987.

[14] B. Selic, G. Gullekson, and P. T. Ward, *Real-time object-oriented modeling*. John Wiley & Sons, 1994.

[15] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *ASME J. of Dynamic Systems Meas. and Control*, vol. 104, pp. 205–211, 1982.

[16] G. Rodriguez, A. Jain, and K. Kreutz-Delgado, "A spatial operator algebra for manipulator modeling and control," *The Int. J. Robotics Res.*, vol. 10, no. 4, pp. 371–381, 1991.

[17] J. C. Cleaveland, "Building application generators," *IEEE Software*, vol. 4, no. 5, pp. 25–33, July 1988.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley Publ. Co., 1994.

[19] M. Fowler and K. Scott, *UML Distilled: A brief guide to the standard object modeling language*, 2nd ed. Addison Wesley Publ. Co., 1999.

[20] J. Sztipanovits and G. Karsai, "Model-integrated computing," *IEEE Computer*, vol. 4, pp. 110–112, 1997.

[21] R. Höpler and P. J. Mosterman, "Model Integrated Computing in robot control to synthesize real-time embedded code," in *Proc. IEEE Conference on Control Applications*, Mexico City, 2001, pp. 767–772.

[22] R. Höpler, "A unifying object-oriented methodology to consolidate multibody dynamics computations in robot control," Ph.D. dissertation, Technische Universität Darmstadt, 2004.