This is a preprint of an article which appeared in
Robotics and Autonomous Systems 57 (2009) 786 – 795

Adequate Motion Simulation and Collision Detection for Soccer Playing Humanoid Robots

Martin Friedmann, Karen Petersen, Oskar von Stryk

Simulation, Systems Optimization and Robotics Group
Technische Universität Darmstadt
Hochschulstr. 10, 64289 Darmstadt, Germany

Abstract

In this paper a humanoid robot simulator based on the multi-robot simulation framework (MuRoSimF) is presented. Among the unique features of the this simulator is the scalability in the level of physical detail in both the robot's motion and sensing systems. It facilitates the development of control software for humanoid robots which is demonstrated for several scenarios from the RoboCup Humanoid Robot League.

Different requirements exist for a humanoid robot simulator. E.g., testing of algorithms for motion control and postural stability require high fidelity of physical motion properties whereas testing of behavior control and role distribution for a robot team requires only a moderate level of detail for real-time simulation of multiple robots. To meet such very different requirements often different simulators are used which makes it necessary to model a robot multiple times and to integrate different simulations with high-level robot control software.

MuRoSimF provides the capability of exchanging the simulation algorithms used for each robot transparently, thus allowing a trade-off between computational performance and fidelity of the simulation. It is therefore possible to choose different simulation algorithms which are adequate for the needs of a given simulation experiment, for example, motion simulation of humanoid robots based on kinematical, simplified dynamics or full multi-body system dynamics algorithms. In this paper also the sensor simulation capabilities of MuRoSimF are revised. The methods for motion simulation and collision detection and handling are presented in detail including an algorithm which allows the real-time simulation of the full dynamics of a 21 DOF humanoid robot. Merits and drawbacks of the different algorithms are discussed in the light of different simulation purposes. The simulator performance is measured and illustrated in various examples, including comparison with experiments of a physical humanoid robot.

Key words: humanoid robots, motion simulation, collision detection

1 Introduction

In a soccer game of autonomous humanoid robots many different humanoid robot motions must be selected online for fast, goal-oriented motions like fast walking, turning and getting up as well as for ball manipulation with the feet and for collision avoiding navigation. High-level control software for soccer playing humanoid robots consists of several modules like image processing, world modeling, behavior control and motion generation, e.g. [1]. A typical (though simplified and generalized) control application is depicted in Fig. 1.

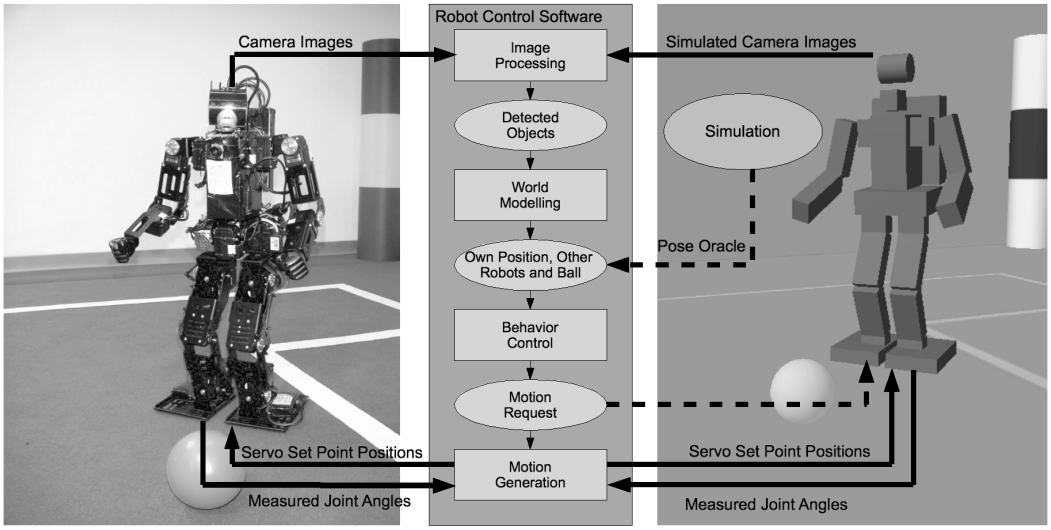

Fig. 1. Data flow in a robot control application. Boxes depict modules of the application, ellipses depict exchanged data. A real robot can only provide information about the sensor data it receives, thus allowing only tests of the whole control software. A simulation of the robot can provide and process further information (dashed arrows) which can be used for testing selected parts of the robot control application.

Testing these modules with the real robot's hardware is not only expensive with respect to time and experimental cost. It is also usually quite difficult to find the reason for an observed robot's misbehavior due to the complexity of its hard- and software. Using offline or "in the loop" simulations of the robot's motion and/or sensing capabilities enables isolated and repeatable investigations of the robot's high-level software and hardware.

Requirements on the physical correctness of the simulation vary for different simulation purposes and scenarios. For successful tests it is crucial to select an *adequate* simulation, which not necessarily needs to be the most accurate one. Testing coordination and control algorithms for a whole team of robots requires high computational efficiency. Testing and optimizing a humanoid robot's motion capabilities requires high physical correctness but no simulation of external sensors.

For testing the self localization of the robot, external sensors like cameras and the

robot's motion must be simulated. As the main concern is not on the sensor processing or motion generation, a sound - but not necessarily too detailed - simulation of these properties is sufficient. Sometimes also a too accurate simulation may even be harmful, as simulated effects like vibration of the robot or blur and noise of the camera may mask errors of the algorithms under observation.

When testing behavior of a team of robots, the main concern is which decisions are made by each robot depending on its knowledge of the world. This information can be generated directly from a simulation without using the image processing and world modeling software modules. Likewise the robots motion capabilities can be simulated in a simplified manner.

In this paper a flexible and modular simulation for humanoid robots is presented which fosters adequate simulation by providing several different algorithms for motion simulation and collision detection. It differs significantly from the majority of robot simulations used currently in the context of the RoboCup Humanoid Robot League. The algorithms have been implemented within the framework `MuRoSimF` [2], which provides means for integration of different simulation algorithms within one simulation framework.

The remainder of this paper is structured as follows: The next section presents an overview on algorithms for robot dynamics simulation, existing simulation packages and simulations. In Section III the structure of simulations and models built with `MuRoSimF` are described. Section IV, V and VI present the algorithms used for motion simulation, collision-detection and handling and sensor simulation which are presently available. Results are presented in Section VII. The paper concludes with a discussion of the results.

2 Overview

2.1 Multi-body System Dynamics

Humanoid robots usually consist of rigid kinematic chains in a tree structure. As postural stability is of outmost importance the robot dynamics must be considered. The inverse dynamics of a multi-body system (MBS) with a fixed base and an open rigid, kinematic chain of n degrees of freedom (DOF) is generally described by the n-dimensional system of nonlinear second order differential equations

$$\tau = M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) + G(\mathbf{q}). \tag{1}$$

In this equation τ is the vector of joint-forces required to yield the acceleration $\ddot{\mathbf{q}}$ in the joints. M is the symmetric and positive definite mass-matrix describing the inertia of the bodies of the system, C are the Coriolis-forces acting onto the joints, G

are the gravitational forces. For bipedal and four-legged robots with varying contact situations and forces, the resulting system of differential algebraic equations can be transformed to a similar second order system using a reduced dynamics approach as described in [3]. Further external forces which may be caused by collisions or friction can be introduced by adding a vector $F(\mathbf{q}, \dot{\mathbf{q}})$ depicting the impact of these external forces on the single joints. This leads to

$$\tau = M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) + G(\mathbf{q}) + F(\mathbf{q}, \dot{\mathbf{q}}). \tag{2}$$

Calculating τ from $\ddot{\mathbf{q}}$ is known as the problem of inverse dynamics. It can be solved efficiently by the well known Recursive Newton-Euler Algorithm (RNEA, e. g. [4]) in $O(n)$ runtime without explicitly calculating M.

Solving the equation for $\ddot{\mathbf{q}}$ (for given τ) is known as calculation of the forward dynamics. Prominent algorithms for solving this problem are the Composite Rigid Body Algorithm (CRBA) [5] and the Articulated Body Algorithm (ABA) [6], cf. [3]. The CRBA explicitly calculates the mass matrix M column wise by evaluating the RNEA $n+1$ times for $\ddot{\mathbf{q}} = \mathbf{0}$ resp. $\ddot{\mathbf{q}} = \mathbf{u}_i$ where $\mathbf{u}_i$ denotes a unit vector where the i-th component is 1. The resulting linear equation is solved leading to an order of $O(n^3)$. Instead of that the ABA solves this equation implicitly yielding an $O(n)$ algorithm. For a small DOF the CRBA will perform better with a break even at a DOF of 6 to 9 (depending on the implementation) [7].

For a reasonable simulation of the robot's dynamics, additional robot data not included in the kinematical data is needed. These include mass, center of mass and inertia tensor for each rigid link and joint of the robot as well as models describing the torques generated by the robot's joint drives.

2.2 Simulation Packages

For simulating the motion of multi-body systems, several packages exist which provide algorithms for numerical integration, collision detection and handling and robot modeling.

The Open Dynamics Engine *ODE* [8] is an open-source-project that provides collision detection and handling for several geometric primitives like box, cylinder and sphere as well as meshes and an algorithm for rigid body dynamics. Each joint of the robot and each collision is modeled as a geometric constraint that removes one or more DOF from the system. ODE provides two methods for solving the system's equation: an accurate solver with $O(n^3)$ runtime complexity and a less accurate iterative solver with $O(m \cdot n)$ runtime complexity (where m is the number of iterations). Numerical integration is done using a first order integrator which was chosen by ODE's developers to emphasize speed and stability over accuracy. To speed up

collision detection ODE supports multi-resolution hash-tables and quad-trees. ODE has been used on several platforms including Linux, Windows and MacOS X.

Bullet-Physics [9] is an open-source package providing rigid-body physics and collision detection for primitive shapes as well as for meshes. To speed up collision detection it makes use of axis aligned bounding boxes. It is available for Linux, Windows and MacOS X platforms.

AGEIA PhysX [10] is a physics engine providing rigid body dynamics and collision detection mainly used in games. It is possible to accelerate the calculations with specialized add-on cards. A software-development-kit is available for Windows, Linux and the XBOX game-console.

The *Newton Game Engine* [11] is a closed-source package for physics simulation and collision detection. The binary libraries are distributed freely for Windows, Linux and MacOS X. It uses a deterministic solver, unlike other physic engines that use iterative algorithms.

Karma [12] is a physics engine developed by Epic Games. Collision detection is only performed with bounding volumes. It can be decided for each body whether the motion simulation should be accurate (but slow) or quick (but not very precise).

SOLID [13] is a collision detection library that is available either under a public license (GPL or QPL), which requires to publish programs using SOLID as open source, or for commercial use for a fee depending on the usage of the library. The collision detection pipeline consists of three phases: the broad phase, where the intersection between axis aligned bounding boxes is calculated, the complex phase, where each complex object is seen as a hierarchy of primitives, and the exact phase, where the intersection of potentially colliding primitives is computed. SOLID supports a lot of different shape types, including boxes, cones, cylinders, spheres, ellipsoids, line segments, triangles, quadrilaterals, general convex polygons, and convex polyhedra. As SOLID only provides collision detection, the collision response has to be added by the user.

GIMPACT [14] is an open source collision detection library that supports concave meshes and deformable bodies. GIMPACT is available under a free license. It is used for mesh-collisions in the Bullet-Physics library.

2.3 Existing Simulators

SimSpark [15] is a simulation framework which is used in the RoboCup 3D simulation league. It uses ODE for physics simulation. New robot models and environments can be added using a description language.

USARSim [16] is a simulation environment based on a the *Unreal Engine* by epic games [17], which provides not only physics simulation, but also tools for visualization and integration of description languages. The physics simulation in Unreal Engine 2 is based on Karma, Unreal Engine 3 uses AGEIA PhysX. With USARSim a variety of different robots can be simulated, including humanoids as shown in [18]. A variety of sensors is provided. A simulator based on USARSim is used in the RoboCup Rescue Simulation league.

SimRobot is a simulator based on ODE. In [19] the simulation of wheeled and four-legged robots was shown. Other robots and different environments can be defined with *RoSiML* [20], a description language based on XML. Many sensors like cameras and different distance sensors can be simulated.

Microsoft Robotics Studio [21] provides tools for programming robots as well as a 3D simulator. The simulation of many different sensor types is available. The dynamics are based on AGEIA PhysX.

Webots [22] is a commercially available simulation that deals with wheeled, legged and flying robots and provides a lot of different sensors. The physics simulation is based on ODE.

Gazebo is the 3D-simulator of the player/stage project [23], [24]. It supports the simulation of cameras and distance sensors. Motion simulation is based on ODE.

OpenHRP [25] is a simulator for humanoid robots based on CORBA. Collision detection and forward dynamics are realized as CORBA-servers, each of them can be exchanged by other implementations.

2.4 Discussion

As mentioned in Sect. 1, different purposes and test scenarios have quite different requirements and needs.

All of the above mentioned simulators rely on some externally developed physics engine for motion simulation. Even though many of these engines provide good performance and accuracy, some problems may arise from this approach. For closed source packages, it is impossible to change the provided simulation algorithms. Often it is not even documented, which kind of algorithm is used. Open source packages obviously enable inspection and adaption of the source code. Nevertheless this is an error prone effort, as often the only available up-to-date documentation is the source-code itself. To the authors best knowledge, none of the packages discussed before provides a flexible exchange of the simulation algorithms used. By choosing one of these packages and developing a simulation based on them, a developer will be stuck to these specific algorithms. Additionally, exchanging one package for an-

other is complicated by the fact that each package uses its own data representation.

To overcome these problems and limitations, MuRoSimF provides a way to combine different simulation algorithms and models. Several algorithms differing in accuracy and run-time requirements are provided. Unlike in simulations which are based on a specific simulation package, it is possible to use the provided algorithms in any combination within one simulation simultaneously. A possible use of this feature is the investigation of one robot's reactions on the actions of a whole team of other robots. To do this, the robot under investigation can be simulated with high accuracy while for the other robots a reduced level of detail can be applied.

3 Structure of Simulation

A simulation consists of two main parts: Models of the systems under consideration (e. g. robots, ball or environment) and simulation algorithms computing the behavior of these models (e. g. motion simulation, collision detection or visualization, an example setup is given in Fig. 2). Within MuRoSimF each model is described as a set of objects where each object contains a set of constant and variable properties describing the object. Constant properties are assigned to an object during creation of the model, variable properties are assigned during simulation setup if they are needed by a simulation algorithm as input or output data.

After setup the simulation algorithms are executed. The rate for each algorithm can be adjusted individually. During execution the simulation can be connected to the robot control software. An interface is provided to run low-level control modules synchronously with the simulation.

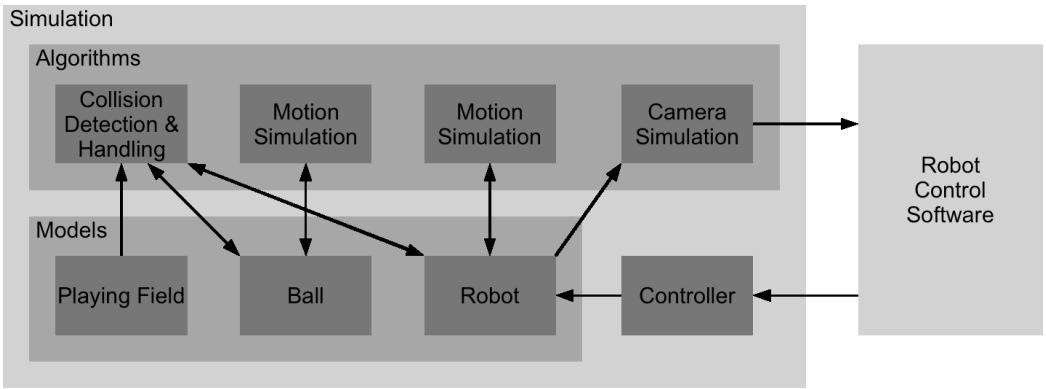

Fig. 2. Structure of a simulation, consisting of models for robot and ball, algorithms for motion and camera simulation and collision detection.

Table 1
Basic objects for modeling the robot's structure.

Object	modeling parameters
Base	no parameters
Rigid translation	length and direction
Rigid rotation	axis and angle
Revolute joint	direction of axis
Prismatic joint	direction
fork	no parameters
endpoint	no parameters

Table 2
Some of the properties which may be added to an object during modeling of a robot.

Property	symbol	type	used by algorithm
mass	m	$\mathbb{R}$	dynamic motion simulation
center of mass	com	$\mathbb{R}^3$	
inertia tensor	I	$\mathbb{R}^{3x3}$	
shape	see Sect. 5		collision detection / visualization
surface parameters			collision handling
color / texture			visualization

3.1 Robot Modeling

Robots are modeled as sets of objects connected in a tree structure. An object essentially is a container holding several constant or variable properties. As shown in [26] it is desirable to have a limited set of basic objects for modeling the robot. In `MuRoSimF` these objects are the robot's base, rigid translations, rigid rotations, revolute joints, prismatic joints, forks and endpoints. Each of these objects basically describes a homogeneous transformation relative to the last link of the kinematic structure. With the exception of endpoint and fork after each object one following object is added to the structure. Forks are used to add two following links. Endpoints describe the end of one limb and thus have no following link.

Using these seven basic objects the robot's kinematic structure is modeled. The objects initially only hold information describing its kinematic structure (see Table 1). When modeling the robot, additional constant properties may be added to each object. Which properties are added strongly depends on the level of detail of the model as well as on the simulation algorithms which are to be used. Examples of available properties are given in Table 2.

Two subsets of the robot's objects are of special interest. The set of *bodies* consists of all objects which can experience external forces. These are all objects with the property *mass* (the object will experience gravity) or *shape* (the object can collide with other objects in the simulation). The set of *joints* consists of all revolute and prismatic joints of the robot.

All properties are stored at the bottom of a three-level hierarchy: The simulation provides access to each robot, each robot provides access to a set of object, each object provides access to a set of properties. It is possible to access each property through this hierarchy for monitoring and manipulation of the simulation. This allows an easy expansion of the simulation with new algorithms.

3.2 Simulation Setup

When setting up a concrete simulation, simulation algorithms have to be chosen and connected to the models of the simulated systems. During setup, the algorithms investigate the models they are connected to and may add additional variable properties (e.g. Table 3) to the model's objects (see Fig. 3).

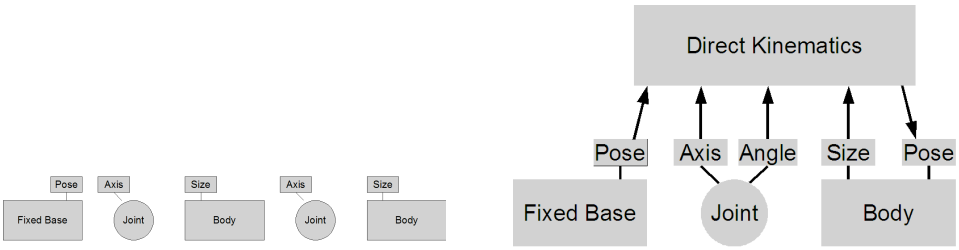

Fig. 3. Left: Modeling of a 1 DOF robot arm. Only the known properties are added to the model. Right: When connecting a simulation algorithm (here: direct kinematics) to the model, additional properties needed or calculated by the algorithm are added to the model.

Table 3
Some of the properties which may be added to an object by a specific simulation algorithm.

Property	symbol	type
Position of Object	$\mathbf{r}$	$\mathbb{R}^3$
Orientation of Object	$\mathbf{R}$	$\mathbb{R}^{3x3}$
Velocity of Object	$\mathbf{v}$, ω	$\mathbb{R}^3$,$\mathbb{R}^3$
Acceleration of Object	$\dot{\mathbf{v}}$, $\dot{\omega}$	$\mathbb{R}^3$,$\mathbb{R}^3$
joint position	q	$\mathbb{R}$
joint rate	$\dot{q}$	$\mathbb{R}$
joint acceleration	$\ddot{q}$	$\mathbb{R}$
external forces and moments	$\mathbf{f_{ext}}$, $\mathbf{n_{ext}}$	$\mathbb{R}^3$

3.3 Execution of Simulation

After creation of the models and setting up the algorithms the simulation may be executed. To do this, a scheduler of all algorithms is set up. Each algorithm can be executed at its own rate, for example 1 ms for motion simulation, but 100 ms for camera simulation.

After setup the schedule is processed by a runtime-system. This runtime-system is used to provide execution of the algorithms, graphics windows and communication with external control software. It is used as an abstraction layer from the underlying operation- and windowing-system. The runtime-system provides means to execute the simulation in soft-real time triggered by the computer's clock. It is also possible to run the simulation faster or slower than real time. If the computation time of the algorithms exceeds the simulated time interval, a warning is issued as real-time simulation is not possible for the given setup.

Algorithms which are to be executed with the same rate and which access disjunct sets of properties may be executed in parallel. First results on the scalability are presented in [27].

For simulations which do not need visualization (e. g. if no cameras need to be simulated), it is also possible to run the simulation without a windowing system and without a timer at the maximum speed of the CPU.

3.4 Integration With Control Software

Two possibilities exist to integrate the control software with the simulation. For higher-level modules (e.g. self localization or behavior control) *asynchronous* inter process communication (IPC) is used to transfer data like motion requests or simulated camera images between simulation and control software. Several communication methods like TCP/IP or (virtual) RS232-connections can be used. Control software integrated by IPC may be run on a different computer than the simulation.

For software modules which are run in hard real-time on a real robot's controller (e.g. motion generation) a different approach is used. This software can be integrated directly into the simulation using dynamic linked libraries. By this it is possible to execute the low-level control functions *synchronously* with the simulation. Control software integrated this way is always executed on the same machine as the simulation.

4 Motion Simulation

To simulate the motion of a robot, several algorithms have been implemented. Each of these algorithms can be used with any humanoid robot modeled using the methodology described above. All algorithms yield information on the motion of the robots base and limbs, but they differ in realism and runtime consumption. The algorithms may be exchanged transparently within `MuRoSimF`. It is therefore possible to choose an algorithm which is most appropriate for a given simulation experiment.

4.1 Kinematic Walking Simulation

This is the most basic algorithm provided for biped motion simulation. It assumes, that the robot is walking on a plane. For each time-step the algorithm calculates the direct kinematics of the robot while keeping the standing foot from the last time-step in a fixed position on the walking plane. If this leads to a configuration of the robot in which the other foot penetrates the plane, the roles of the feet are swapped (see Fig. 4).

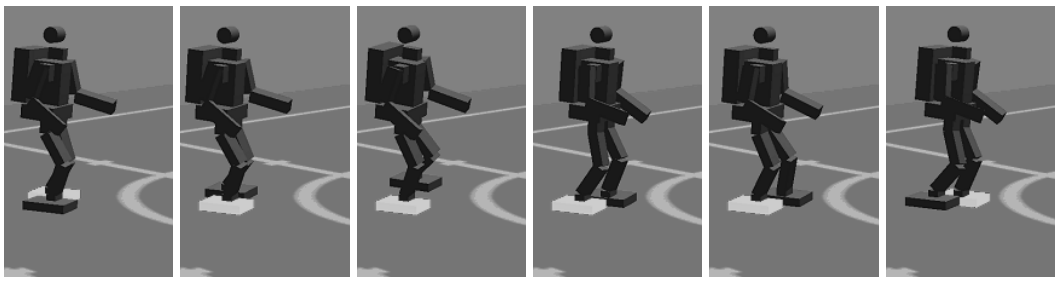

Fig. 4. Kinematic walking simulation of a humanoid robot. The current standing foot is marked yellow.

Obviously this algorithm will produce sound results only if proper walking motions are executed. The algorithm can be used to evaluate sensing abilities of the robot which are influenced by the robots motion (e. g. the changing pitch of the camera). It is also useful for testing behavior control for teams of robots as it is computationally cheap.

As the algorithm does not consider the dynamic behavior of the robot's servo motors, there is no need for simulating the servo's properties. Instead the desired position is used as current position of the joint.

When using the algorithm it is not possible for a simulated robot to fall over. This quality of the algorithm can be very helpful when performing the experiments mentioned above, as wobbling of the robot or other disturbances may mask faulty be-

havior of the algorithms under consideration. If postural stability or other motions beyond slow walking are of interest, another algorithm should be chosen.

4.2 Simplified Dynamics Simulation

To allow the robot to perform motions beyond slow walking in a more realistic manner a simplified dynamics simulation algorithm has been developed. This algorithm uses the common center of mass and inertia tensor of the robot. This information is used to calculate the motion of the robot's base depending on external forces the robot experiences by gravity or contact. To do this, all external forces are summed up and transferred to the robots center of mass, where the dynamics are calculated for the robot as if it was a single rigid body.

Just like the kinematic walking simulation this algorithm only needs information on the current position of the servos. Only if frictional forces are to be introduced to the system by the collision handling algorithm (see Sect. 5.3), it is necessary to take the rate of the joints into account as well.

The algorithm can be used to simulate a wide variety of motions including falling down and getting up. As the robot's feet are not fixed to the plane, the algorithm will generate some shaking of the simulated humanoid robot.

Even though the algorithm yields sound results for many types of motion (e.g. Fig. 11), it is not a simulation of the full MBS dynamics of the robot. As the algorithm does not consider the reactive forces caused by motion of the robot's joints, it cannot be used to simulate any effects based on these forces like balancing.

4.3 Full Dynamics Simulation

To overcome the limitations of the simplified dynamics simulation a full dynamics simulation has been developed.

The algorithm considers the positions, velocities and accelerations of a given trajectory of the robot's joints. These are calculated from the desired positions provided by the motion generation software under the assumption that the desired position is reached within one control cycle.

By this a simulation of the individual servos is avoided and the forward dynamics problem is reduced to six degrees of freedom, as the only accelerations not known are those for the free base of the robot. These are calculated by an adaption of the CRBA which only considers the first six rows of Eq. (2), yielding the upper left 6×6 submatrix $\tilde{M}$ of M as well as the first six components $\tilde{F}$ of $(C(\mathbf{q}, \dot{\mathbf{q}}) +$

$G(\mathbf{q}) + F(\mathbf{q}, \dot{\mathbf{q}})$). As the robot's base is free, it does not experience any forces, so that the accelerations of the base can be calculated by solving

$$\mathbf{0} = \tilde{M}\ddot{\mathbf{q}} + \tilde{F}. \tag{3}$$

If the acceleration of the base is calculated this way, all internal reacting forces caused by accelerations of the joints are considered as well as any external forces from the term $F(\mathbf{q}, \dot{\mathbf{q}})$. This leads to a more realistic simulation of the robot's motions, as effects like balancing can be displayed. In combination with a simulation of inertial sensors this property can be used for testing stabilization algorithms like the one presented in [28].

Obviously this algorithm is not able to simulate any effects caused by the robot's servos like limited torque of the motor or tolerance of the gear. These limitations can be overcome by using a dynamics algorithm considering the torque of the motors (e.g. the ABA) in combination with an appropriate model of the servos.

5 Collision Detection and Handling

The simulated scene contains several sets of objects, called compounds. A compound, e.g., consists of all objects belonging to a robot or all static objects in the scene. To speed up collision detection, each compound is organized in a bounding volume hierarchy. Currently, spheres are used as bounding volumes, but due to the modularity of the framework, other bounding volumes like oriented bounding boxes (OBBs), axis aligned bounding boxes (AABB), capsules, etc., can easily be added. It is a well-known fact that the choice of the bounding volume is always a tradeoff between accuracy (fitting of the bounding volume) and complexity (computational effort for the intersection test and possibly re-calculation after rotation, memory usage) [29]. The flexibility of `MuRoSimF` allows to select the appropriate bounding volume for each simulation, to make sure that the hierarchy is best adapted to the current setup. To improve performance the detection of inner collisions can be disabled for each compound separately.

Two classes of compounds exist: Unstructured compounds are used to store arbitrary sets of objects while structured compounds are used to store sets of objects having a defined relation (e.g. tree structured robots).

5.1 Building the Tree

For the sake of simplicity binary trees are used. Only the leaves represent bodies of the robots. All inner nodes store the bounding volume for their succeeding subtree.

For unstructured compounds a top-down-approach is used, that divides a set of objects into two disjunct sets in each step. The iteration is stopped, when each node contains one single object. With this method the resulting trees automatically fulfill the requirements mentioned above. In the RoboCup simulation, the playing-field is the sole unstructured compound. As it is static, the hierarchy has to be calculated only once during initialization.

Each robot is a structured compound. The structure described in Sect. 3.1 has to be transformed to meet the requirements mentioned above. From a robot's compound (see Fig. 5) only the *bodies* are needed for collision detection. As first step every node without a physical body is replaced by an empty node. Then empty nodes with less than two successors are deleted (e. g. the camera-node and the joint-nodes in Fig. 5). After this transformation, the structure has been reduced a lot, but there are still some nonempty inner nodes, e.g. the base node in Fig. 5. These nonempty nodes are moved to the leaves by inserting additional empty inner nodes (see lower part of Fig. 5).

As the structure of a robot does not change, the tree has to be calculated once, only the bounding volumes have to be recalculated after the robot has moved.

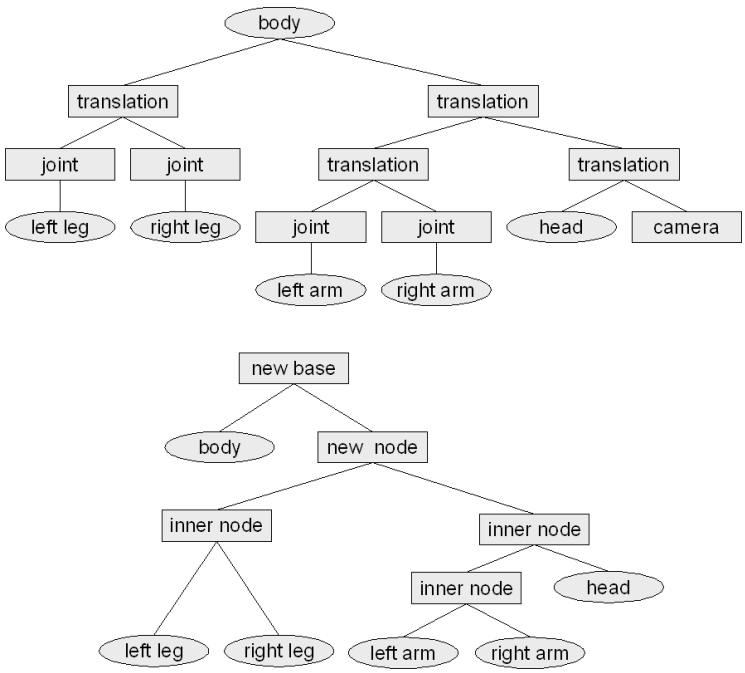

Fig. 5. Top: Tree-structure of a simplified humanoid robot, consisting of a body, two arms with one joint each, two legs with one joint each and a head with a camera. Bottom: The transformed tree that fulfills the requirements for collision detection.

5.2 Intersecting Two Trees

To detect collisions of two compounds, only the respective trees have to be intersected. There are several approaches for intersecting two trees. If the bounding volumes have an intersection it is possible to descend simultaneously in both trees, to descend first completely in the tree with the bigger bounding volume, or to descend in the tree that currently has the bigger bounding volume.

When intersecting a robot with the environment, it is not reasonable to descend in both trees simultaneously, as the playing field is much bigger than a robot. When intersecting two robots of the same size, it usually takes more time to first descend completely in the tree that seems to be bigger. Descending always in the tree with the currently bigger bounding volume requires a comparison of the sizes in each step. As this can be determined easily by comparing the radii of the spheres this approach is taken for the regarded application.

5.3 Collision Response

To calculate the resulting forces of a collision, a soft contact model is used, which allows the bodies to penetrate each other. The collision depth c_{depth} and the collision normal $\mathbf{c}_n$ calculated by the collision detection are used for calculating the normal force

$$\mathbf{f}_n = \begin{cases} 1 \cdot s_c \cdot c_{depth} \cdot \mathbf{c}_n & \text{(objects getting closer)} \\ s_b \cdot s_c \cdot c_{depth} \cdot \mathbf{c}_n & \text{(objects separating)} \end{cases} \tag{4}$$

that uses a spring model with different spring constants s_c and a scaling factor s_b. The friction $\mathbf{f}_{fric}$ between two colliding bodies is calculated depending on the relative linear velocity and the constant s_μ which is defined for each pair of surface-materials. As there is only one contact point an additional pseudo friction $\mathbf{n}_{fric}$ depending on the relative angular velocity is calculated

$$\begin{aligned} \mathbf{f}_{fric} &= \mathbf{v}_{\mathbf{rel}} \cdot s_\mu \cdot \mathbf{f}_{\mathbf{n}} \\ \mathbf{n}_{fric} &= \omega_{\mathbf{rel}} \cdot s_\nu \cdot \mathbf{f}_{\mathbf{n}} \,. \end{aligned} \tag{5}$$

6 Sensor Simulation

For closed loop testing of the robot control software a simulation of the robot's sensing devices or an adequate replacement is necessary. In this section several possibilities for sensor simulation provided by `MuRoSimF` are discussed.

6.1 Sensors Providing Scalar Values

Several sensors with scalar values like inertial sensors, joint position encoders or contact sensors are applied in humanoid robots. Inertial sensors like gyroscopes, accelerometers or joint position encoders can be simulated, as their respective values are provided by the motion simulation. Contact sensors can be simulated using data from the collision detection which provides information, if a contact has occurred. Additionally contact forces can be simulated using the collision handling module.

A common property of all of these sensors is, that properties (e.g. the acceleration) of a simulated object are mapped to a scalar value. For the development of new sensor simulations a common base class is provided. When deriving new simulations from this base class, only the mapping from the respective property to a scalar value must be implemented. The base class already provides commonly used post processing functions like simulation of sensor saturation, random noise and simulated A/D conversion which can be adapted to the newly derived sensor simulation.

6.2 Cameras

The main external sensor for robots playing soccer under the rules of the RoboCup Humanoid League are cameras. The camera simulation uses the visualization subsystem of `MuRoSimF`, which is based on OpenGL real-time rendering.

After rendering the scene from the camera's point of view, the image can be post-processed to reproduce some features of the camera (see Fig. 6). The image can be blurred using a Gaussian filter and it is possible to simulate the distortion caused by the lens of the camera. The latter is done by measuring the distortion of the real camera's lens using the well known Camera Calibration Toolbox for Matlab [30] and applying the same distortion to the image. The lens is described by focal center, focal length and a sixth order polynomial for the distortion of a view ray depending on its direction.

As the simulated camera images are easier to process than images from real cameras, the simulation's main use is to test and debug the robot's image processing software under optimized conditions. The simulation of the lens distortion has proven very helpful when debugging the software module used to calculate view rays to detected objects.

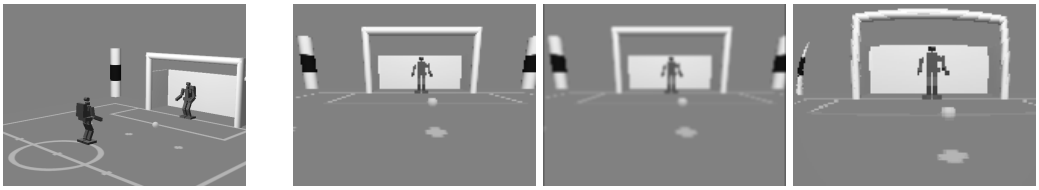

Fig. 6. Simulation of a camera with an aperture angle of $85\,\mathrm{deg}$, mounted in the upper chest of a humanoid robot. Left: the simulated scene. Left middle: plain image from the camera. Right middle: same image with gaussian blur. Right: same image with lens distortion.

6.3 Testing Without External Sensors

When debugging the behavior module of a robot's control software, it is often desirable to avoid errors caused by image processing or self localization which can mask errors of the behavior control. To do this kind of testing, it is possible to send ground truth of position and orientation of all robots and the position of the ball within the simulation to the robot control software.

7 Results

The simulation algorithms presented in this paper have been used successfully in testing the control software for a 21 DOF kid-size humanoid robot used by the authors' team in the RoboCup 2006 and 2007 [1] competitions.

7.1 Testing of World Modeling

All decisions of the robot's autonomous behavior are based on its knowledge of its environment, which is provided by the world modeling module. The simulation has been used in testing the self-localization (see Fig. 7) and ball modeling modules. The modules' quality is evaluated by comparing their results with the "true" values provided by the simulation.

Using such tests several sources of error (e.g. blur of camera, changes in lighting or jiggling of the robot) which happen in normal operation of the hardware can be avoided. By this it is much easier to track errors in the algorithms being monitored.

7.2 Testing of Behavior Control

The simulation has been proven very useful when testing the robot's behavior. At first a behavior is tested by using the position of robots and ball provided by the sim-

Fig. 7. Interactive testing of self localization. Left window: Interactive simulation. Right window: The data provided by the self-localization software (red arrows depict particles, dark blue arrow is pose estimate) can be compared to the robot's position and direction in the simulation (sole blue arrow).

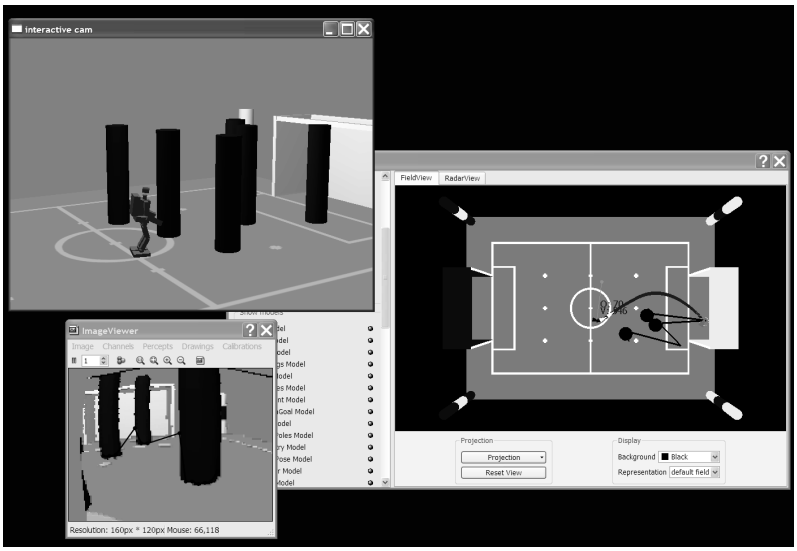

Fig. 8. Testing the behavior for the obstacle challenge. Upper left window: interactive simulation. Lower left window: simulated camera-image. Right window: debug information from the control software: black circles depict detected obstacles, the red curved arrow is the currently planned collision free way to the goal.

ulation, thus avoiding any potential complications caused by the computer-vision or world-modeling modules of the software. If this test is successful, the behavior can be tested with simulated camera images (see Fig. 8). As the simulation provides the possibility to repeat experiments under exactly the same circumstances debugging of unexpected behavior is facilitated very much. The simulation is not limited to single robots, but can be used to test the behavior of whole teams of robots, too (see Fig. 9).

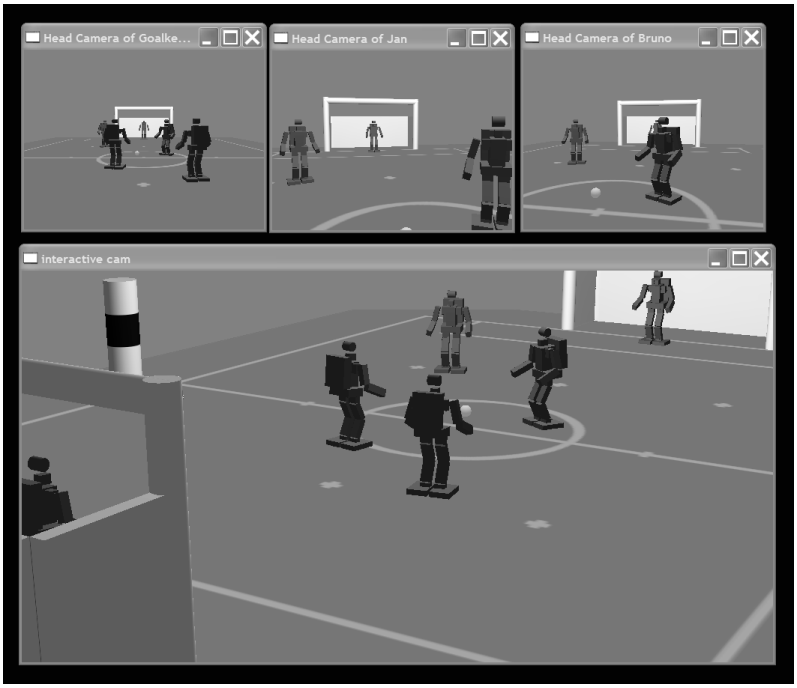

Fig. 9. Testing of three player team behavior. Only the cameras for the team under consideration (the robots watching the yellow goal) are simulated in this setup.

7.3 Quality of Motion Simulation

With the dynamic algorithms presented in Sect. 4 the motions of the simulated robot are comparable to the real robot's motions. This especially includes falling and get-up motions (see Fig. 10 and Fig. 11) which cannot be simulated correctly by the kinematic walking simulation.

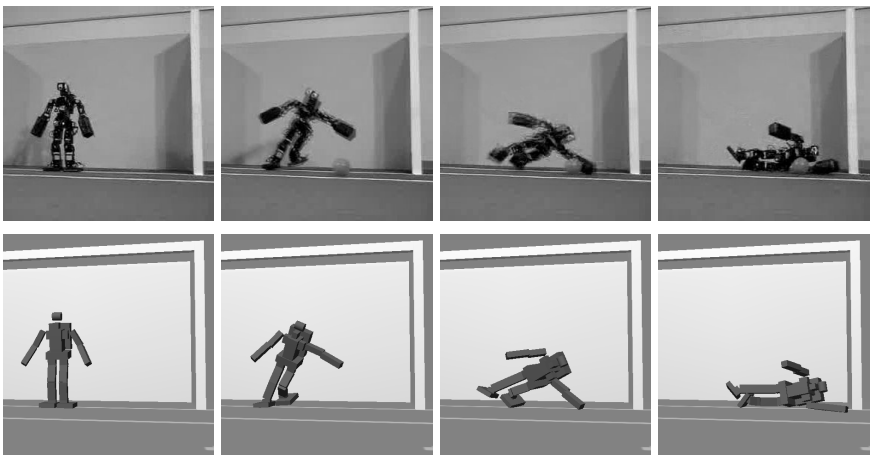

Fig. 10. Top: Goal keeper motion on the real robot. Bottom: the same motion simulated with the simplified dynamics algorithm.

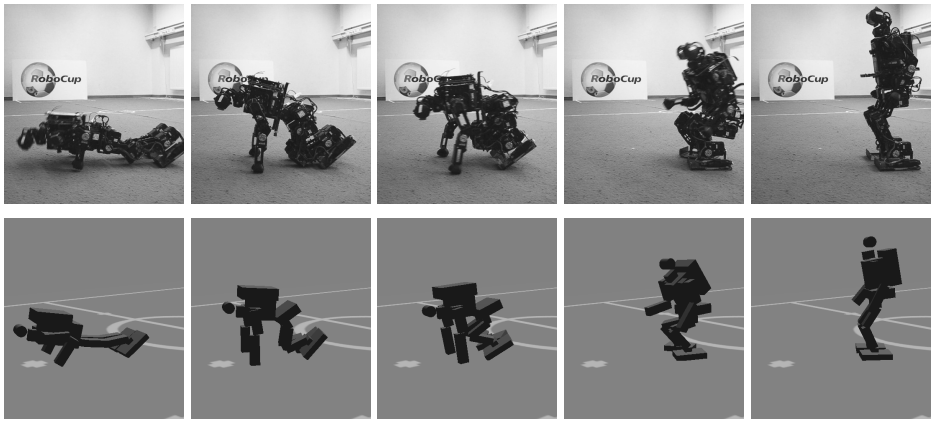

Fig. 11. Top: the real robot while getting up. Bottom: the same motion simulated with the simplified dynamics algorithm.

7.4 Evaluation of New Hardware

Besides testing of software for existing hardware the simulation also can be used for evaluating alternative robot hardware components or environments. For example, the simulator can be used to evaluate several possible camera configurations (opening angle, resolution) to meet the requirements of different tasks or environments (as a greatly enlarged playing field for humanoid soccer in RoboCup 2008).

As the algorithms for dynamics simulation are not limited to biped robots, the simulation also has been used during the design of a new four-legged robot platform [31]. By this it was possible to optimize the design of the robot's neck for viewing capabilities. First experimental gaits could be developed before the hardware was available, thus reducing the development time.

7.5 Performance of Simulation

Due to the introduction of the newly developed collision detection (see Sect. 5), the performance of the simulation could be improved significantly compared with the data given in [2]. In an experimental setup the collision of a model-car (seven bodies) with ten 21-DOF humanoid robots (each 20 bodies) (see Fig. 12) could be simulated in real time using the simplified dynamics algorithm (with a rate of 1000Hz) on a standard laptop computer (Intel Pentium M CPU (1.86GHz), 1GB of RAM, ATI mobility Radeon X700 graphics chip set). On the same computer 6 robots, each equipped with a camera on 20 fps, could be simulated in real-time using kinematic walking.

To measure the absolute performance, the intersection tests in each time-step were counted. Simulating the RoboCup field with two robots, without using the bounding sphere hierarchy, requires 973 intersection tests in each time-step. With the

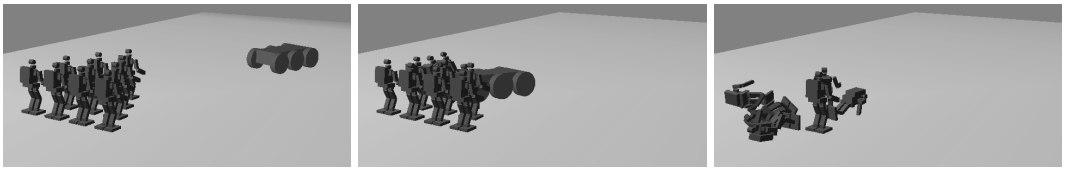

Fig. 12. Test scenario: collision of a model-car with ten humanoid robots.

collision detection described in Sect. 5, depending on the configuration of the movable objects, on average 178 intersection-tests between bounding spheres and 45 object-intersection tests are performed per time-step. The distribution can be seen in Fig. 13. This shows, that on average there have to be performed only 23% of the initial tests per time-step. Most of them are intersection tests between spheres which are very fast to calculate. Even in the worst case only about 40% of the tests have to be performed, compared to the collision detection presented in [2].

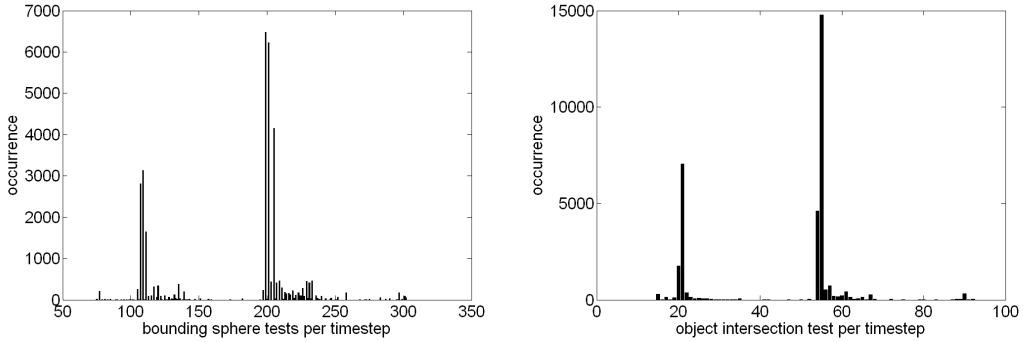

Fig. 13. Left: Bounding sphere tests that have to be performed during a simulation with two robots. Right: Object intersection tests that have to be performed in the same scenario.

8 Conclusions and Outlook

In this paper a humanoid robot simulator based on the multi-robot simulator framework `MuRoSimF` has been presented which enables adequate motion simulation for different needs (e.g. physical accurate simulation for testing a robot's motions or efficient simulation of larger teams for testing autonomous behavior, communication and cooperation) without the necessity to model the robot several times for different simulators. A kinematic walking simulation and a simplified dynamics simulation were used for testing behaviors. In addition an algorithm for full dynamic simulation of a multi-body system was discussed. The simulator is freely available for other researchers for non-commercial research purposes from the authors.

The performance of the simulation could be highly improved compared to [2] with the help of bounding sphere hierarchies. In that way a collision between a model

car and 10 humanoid robots could be simulated in real-time on a standard laptop. Real-time simulation for six robots equipped with one camera each is possible thus allowing tests for 3-on-3 games. Several videos of humanoid robot simulations can be found at www.dribblers.de/murosimf.

One of the central features of the underlying `MuRoSimF` is the decoupling of the simulation's data model and the algorithms used for the simulation. This allows an easy expansion of the simulation with new algorithms. The next steps in the development of `MuRoSimF` will be the integration of full multi-body-systems dynamics and the expansion of the collision detection system to arbitrary convex shapes represented by triangle meshes.

Integration of these algorithms will help to improve the simulation towards more physical realism. An obvious next step is validating the simulation and the models it uses by comparing the performance of real and simulated robots. One central tool for validation will be well defined test-sets (like the ones used for vehicle physics in [32]). In the context of the simulator described in this paper, it will be necessary to consider the differing levels of physical detail by providing differing test-sets for the differing levels of detail. For validation of the physical correctness of the robot dynamics and motor models used by means of experimental data, approaches like presented in [33] can be used as it has been done for quadruped robots and will be adapted to the models presented in this paper.

The hierarchically structured data model of the simulation allows easy access to any model information used or calculated by the simulation. A plugin-interface allowing access to this data is currently under development to allow easy integration of user defined code for control and monitoring of the simulation. This interface will be used to integrate a scripting engine into the simulation to allow automation of the simulation.

Acknowledgment

Parts of this research have been supported by the German Research Foundation (DFG) within the Research Training Group 1362 "Cooperative, adaptive and responsive monitoring in mixed mode environments".

References

[1] M. Friedmann, J. Kiener, S. Petters, D. Thomas, O. von Stryk, Reusable Architecture and Tools for Teams of Lightweight Heterogeneous Robots, in: Proc. 1st IFAC Workshop on Multivehicle Systems, Salvador, Brazil, 2006, pp. 51–56.

[2] M. Friedmann, K. Petersen, O. von Stryk, Tailored Real-Time Simulation for Teams of Humanoid Robots, in: U. Visser, et al. (Eds.), RoboCup 2007: Robot Soccer World Cup XI, Vol. 5001 of Lecture Notes in Computer Science/Artificial Intelligence, Springer-Verlag, Berlin/Heidelberg, 2008, pp. 425–432.

[3] M. Hardt, O. von Stryk, Dynamic modeling in the simulation, optimization, and control of legged robots, Z. Angew. Math. Mech. 83 (10) (2003) 648–662.

[4] J. Luh, M. Walker, R. Paul, On-Line Computational Scheme for Mechanical Manipulators, Transactions of the ASME Journal of Dynamic Systems, Measurement and Control 102 (2) (1980) 69–76.

[5] M. W. Walker, D. E. Orin, Efficient Dynamics Computer Simulation of Robotic Mechanisms, Journal of Dynamic Systems, Measurement, and Control 104 (1982) 205–211.

[6] R. Featherstone, The Calculation of Robot Dynamics using Articulated-Body Inertias, Intl. J. of Robotics Res. 2 (1) (1983) 13–30.

[7] R. Featherstone, D. Orin, Robot Dynamics: Equations and Algorithms, in: Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA), San Francisco, CA, USA, 2000, pp. 826–834.

[8] R. Smith, ODE - Open Dynamics Engine, www.ode.org (2008).

[9] Bullet Physics Library, website http://bulletphysics.com (2008).

[10] NVIDIA PhysX website, http://www.nvidia.com/object/nvidia_physx.html (2008).

[11] Newton website, http://www.newtondynamics.com/ (2007).

[12] Epic Games, Karma, http://udn.epicgames.com/Two/KarmaReference.html (2007).

[13] SOLID Collision Detection Library, website http://www.dtecta.com (2008).

[14] GIMPACT Geometric tools for VR, website http://gimpact.sourceforge.net (2008).

[15] O. Obst, M. Rollmann, Spark – A Generic Simulator for Physical Multi-agent Simulations, Computer Systems Science and Engineering 20 (5).

[16] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, C. Scrapper, USARSim: a Robot Simulator for Research and Education, in: Proc. of the 2007 IEEE Intl. Conf. on Robotics and Automation (ICRA), 2007, pp. 1400–1405.

[17] Epic games, unreal engine, www.epicgames.com (2007).

[18] N. Greggio, G. Silvestri, S. Antonello, E. Menegatti, E. Pagello, A 3D Model of Humanoid for USARSim Simulator, in: First Workshop on Humanoid Soccer Robots, 2006, pp. 17–24.

[19] T. Laue, K. Spiess, T. Röfer, SimRobot - A General Physical Robot Simulator and its Application in RoboCup, in: A. B. et al. (Ed.), RoboCup 2005: Robot Soccer World Cup IX, No. 4020 in Lecture Notes in AI, Springer, 2005, pp. 173–183.

[20] RoSiML website, http://www.informatik.uni-bremen.de/spprobocup/RoSiML.html (2005).

[21] Microsoft Robotics Studio, msdn.microsoft.com/robotics/ (2007).

[22] O. Michel, Cyberbotics Ltd. - Webots(tm): Professional Mobile Robot Simulation, Intl. Journal of Advanced Robotic Systems 1 (1) (2004) 39–42.

[23] Player/Stage Project website, http://playerstage.sourceforge.net/ (2006).

[24] B. P. Gerkey, R. T. Vaughan, A. Howard, The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems, in: Intl. Conf. on Advanced Robotics (ICAR), Coimbra, Portugal, 2003, pp. 317–323.

[25] H. Hirukawa, F. Kanehiro, S. Kajita, OpenHRP: Open Architecture Humanoid Robotics Platform, in: Robotics Research: The Tenth International Symposium, Vol. 6/2003, Springer Berlin / Heidelberg, 2003, pp. 99–112.

[26] R. Höpler, A Unifiying Object-oriented Methodology to Consolidate Multibody Dynamics Computations in Robot Control, No. 1054 in Reihe 8: Meß- Steuerungs- und Regelungstechnik, VDI Verlag, Düsseldorf, Germany, 2005.

[27] M. Friedmann, K. Petersen, O. von Stryk, Simulation of Multi-Robot Teams with Flexible Level of Detail, in: Proc. of Simulation, Modeling and Programming for Autonomous Robots (SIMPAR), LNAI, Springer, Venice, Italy, 2008, pp. 29–40.

[28] M. Friedmann, J. Kiener, S. Petters, H. Sakamoto, D. Thomas, O. von Stryk, Versatile, high-quality motions and behavior control of humanoid robots, International Journal of Humanoid Robotics 5 (3) (2008) 417–436.

[29] C. Ericson, Real-Time Collision Detection, Morgan Kaufmann Series in Interactive 3D Technology, Morgan Kaufmann Publishers, 2005.

[30] J. Bouguet, Camera Calibration Toolbox for Matlab, www.vision.caltech.edu/bouguetj/calib_doc/.

[31] M. Friedmann, S. Petters, M. Risler, H. Sakamoto, D. Thomas, O. von Stryk, A New, Open and Modular Platform for Research in Autonomous Four-Legged Robots, in: K. Berns, T. Luksch (Eds.), Autonome Mobile Systeme 2007, Informatik aktuell, Springer Verlag, Kaiserslautern, Germany, 2007, pp. 254–260.

[32] C. Pepper, S. Balakirsky, C. Scrapper, Robot Simulation Physics Validation, in: Proc. of the Performace Metrics for Intelligent Systems (PERMIS) Workshop, Galthersburgh, USA, 2007, pp. 97–104.

[33] M. Stelzer, M. Hardt, O. von Stryk, Efficient Dynamic Modeling, Numerical Optimal Control and Experimental Results for Various Gaits of a Quadruped Robot, in: CLAWAR 2003: 6th International Conference on Climbing and Walking Robots, Catania, Italy, 2003, pp. 601–608.