Tailored Real-Time Simulation for Teams of Humanoid Robots

Martin Friedmann, Karen Petersen, and Oskar von Stryk

Simulation and Systems Optimization Group
Technische Universität Darmstadt
Hochschulstr. 10, D-64289 Darmstadt, Germany
{friedmann,petersen,stryk}@sim.tu-darmstadt.de
http://www.sim.tu-darmstadt.de

Abstract. Developing and testing the key modules of autonomous humanoid soccer robots (e.g., for vision, localization, and behavior control) in software-in-the-loop (SIL) experiments, requires real-time simulation of the main motion and sensing properties. These include humanoid robot kinematics and dynamics, the interaction with the environment, and sensor simulation, especially the camera properties. To deal with an increasing number of humanoid robots per team the simulation algorithms must be very efficient. In this paper, the simulator framework `MuRoSimF` (Multi-Robot-Simulation-Framework) is presented which allows the flexible and transparent integration of different simulation algorithms with the same robot model. These include several algorithms for simulation of humanoid robot motion kinematics and dynamics (with $O(n)$ runtime complexity), collision handling, and camera simulation including lens distortion. A simulator for teams of humanoid robots based on `MuRoSimF` is presented. A unique feature of this simulator is the scalability of the level of detail and complexity which can be chosen individually for each simulated robot and tailored to the requirements of a specific SIL test. Performance measurements are given for real-time simulation on a moderate laptop computer of up to six humanoid robots with 21 degrees of freedom, each equipped with an articulated camera.

1 Introduction

Besides suitable hardware the performance of an autonomous humanoid robot is mainly governed by the software modules applied for perception, cognition, behavior and motion control. Efficient performance measuring and debugging of these modules on the real robot hardware is very difficult in general. This is due because physical robots are expensive and only limitedly available and may suffer from many experimental tests. At least equally important is that an observed, undesired robot performance can be caused by any of the many interacting software modules used, which in physical experiments are difficult to test in isolation. Replacing the robot by a real-time simulation of its main physical characteristics including the kinematics and dynamics of humanoid motion with many actuated joints enables SIL-testing, monitoring and debbuging of software modules under repeatable and controllable conditions.

In this paper a real-time simulator for teams of humanoid robots is presented which is closely integrated with the robot control framework `RoboFrame` [1]. The simulation can transparently be interfaced with all software modules (see Fig. 1). Many robot designs include controller hardware for motion generation, e.g., [2]. The firmware of the controller can also be integrated into the simulation.

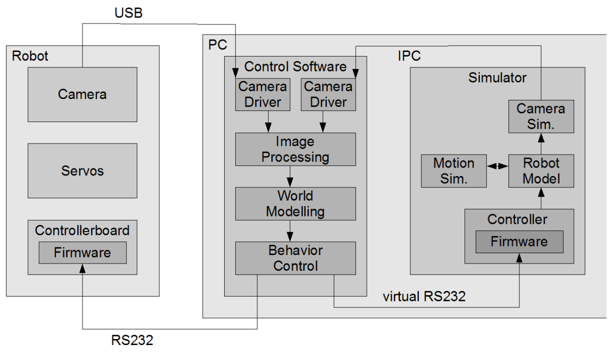


Fig. 1. The modules of the robot control software may be connected transparently either to the real robot or a real-time simulation.

While SIL-tests of humanoid robot motion generation or image processing require a high level of physical accuracy, different requirements exist for testing the behavior control: When investigating how the performance of the behavior control depends on different effects as exact or inexact localization, computer vision with ideal or blurred camera images or walking without or with possible slipping of the feet the simulator must be able to support different levels of simulation detail. Thus, different approaches for behavior-based humanoid robot control can be investigated in an optimal manner by controlling the representations between stimulus and action.

To meet these requirements the simulator must support a flexible exchange and combination of different simulation algorithms for the same purpose, e.g., robot dynamics, on different levels of detail. When exchanging simulation algorithms there should be no need to change the models of the robots or the environment. The simulator framework presented in this paper fulfills these requirements. Several algorithms for real-time motion-, contact- and sensor-simulation are considered. Results for real-time simulation of soccer playing humanoid robots are presented.

2 Overview

2.1 Modeling and simulation of robot motion

Humanoid and four-legged robots are modeled as tree-structured kinematic chains of (usually rigid) links and (usually rotational) joints. The forward kinematics model describes the 3D position and orientation of the robot's bodies depending on the current joint angles. To build the kinematical model the geometrical data of the robot (link lengths, type and position of joints etc.) is needed.

A physics-based modeling of legged locomotion describes the nonlinear relationships between the forces and moments acting at each joint and the feet etc. and the position, velocity and acceleration of each joint angle. The high dimensional nonlinear multibody system dynamics (MBS) results in second order differential equations which can be formulated in various ways differing in terms of efficiency, modularity and flexibility [3, 4]. In addition to the geometrical data a dynamics model requires kinetical data as mass, center of mass and inertia matrix for each link and joint, max/min motor torques and joint velocities which are difficult to obtain and are an often overlooked source of simulation inaccuracy. To simulate interaction with the environment detection and handling of collisions as well as suitable models of foot-ground contacts are required.

In the context of simulation of autonomous robots and RoboCup most often the open source Open Dynamics Engine (ODE) [5] is applied. ODE provides collision detection for several geometric primitives and a simulation of MBS dynamics. It is, however, not documented which method is used to formulate the full or approximative robot MBS dynamics. Only a one-step integrator with constant time step length and without integration error monitoring is available.

2.2 Simulation of sensors

For closed loop simulation it is necessary to provide simulated readings of the robot's internal and external sensors. While joint position sensors can be simulated using a robot kinematics model, simulation of inertial or contact-force sensors requires a dynamics model. The most important external sensors for humanoid robots are cameras. Cameras can be simulated using real-time rendering based on OpenGL or Direct3D. Only few robot simulators (cf. Sect. 2.3, e.g., [6, 7]) enable the simulation of projections beyond the standard pinhole model.

2.3 Overview of existing robot simulators

UCHILSIM [8] is a simulator for the RoboCup Four-legged League limited to the AIBO robots. The motion simulation is based on ODE. OpenGL is used for visualisation and simulation of camera images.

SimRobot [6] is a general simulator for different mobile robots. The current version uses ODE for motion simulation. Simulation of four legged and wheeled robots have been demonstrated in [6]. Simulated sensors consist of tactile sensors, distance sensor and cameras, the later using accelerated hardware rendering.

USARSIM [9] has evolved from a simulator for wheeled systems to a general simulator including legged robots [10]. It provides a wide variety of sensors and includes the simulation of noise. The simulation is based on the proprietary game engine Karma providing a physics simulation and realistic visualization through high accuracy rendering.

Gazebo [11] is a general 3D multi-robot simulator with graphical interface and dynamics simulation. It is part of the Player/Stage project [12]. Motion simulation is based on ODE. Sensor simulation is provided for several sensor systems including distance sensors and cameras. Additional sensors can be added.

Webots [13] is a commercially available general robot simulation package providing a wide variety of legged and wheeled robots. The motion simulation is based on ODE and the visualization uses OpenGL. Webots provides several types of sensors including distance and contact sensors as well as cameras.

2.4 Discussion

All robot simulators mentioned above use either ODE or a proprietary engine as motion simulator which all origin from physics engines in games. In games, however, only a physically plausible appearance based on photorealistic animation and simplified robot dynamics is required and not a quantitavely accurate simulation of moments and forces acting in each of the many humanoid robot's joints. The latter would require a full robot MBS dynamics model as well as error monitoring integration methods. Furthermore, an adaption of the level of detail in robot motion simulation or a flexible exchange or combination of different motion simulators for different robots in the same environment is not possible.

`MuRoSimF` overcomes this problem by introducing a flexible interface allowing the combination of different simulation algorithms. By using the concept presented in this paper, simulation algorithms can be recombined easily for different simulators while maintaining efficient exchange of data.

3 Simulator framework: concept and implementation

3.1 Modeling and integration of algorithms

The simulated robots as well as the environment are described as collections of objects. Each of these objects is a collection of constant properties (like mass, size, shape, etc.) and variable properties (like position, velocity, etc.) which are assigned at runtime to the objects. During creation of robots and environment only the constant properties are assigned. The objects are linked to the algorithms used for simulation next. In this phase, variable properties are assigned to the objects if they are needed by an algorithm. When linking algorithms to an object, it is checked if the object provides the necessary constant properties thus avoiding unnecessary calculation.

Robots are modeled as kinematic trees, consisting of one base, forks, static and variable translations, and static and variable rotations. Each of these elements is a specialized object having the necessary properties like length for a translation or angle and axis for a fixed rotation. During creation of the kinematic tree, additional constant properties may be added to the object, depending on the desired level of detail. All objects belonging to one robot are stored in one container, providing access to several subsets of the objects as the *joints* (denoting all variable translations and rotations) or the *bodies* (denoting all objects which may interact physically with the environment).

3.2 Simulation of humanoid robot motion

Mainly two alternative algorithms for humanoid robot motion simulation are currently utilized. Both have $O(n)$ computational complexity, with n the number of joints/bodies of the humanoid robot and can be individually selected for different humanoid robots in a multi-robot simulation.

Kinematic walking is a computationally cheap algorithm based on the assumption that during walking always (at least) one foot of the robot is in contact with the ground. During each time-step of the numerical integration, the direct kinematics of the humanoid robot is computed with the constraint, that the stance foot is not tilted or lifted off the ground unless the swinging foot touches or would penetrate the ground plane. Then, the roles of the feet are exchanged. Besides its low computational complexity this algorithm has the benefit that the simulated robot can not fall over. If highly accurate simulation of the humanoid robot motion is not crucial, this algorithm can be used successfully, e.g., for testing behavior-based control or self localization modules. The algorithm is restricted to purely humanoid walking applications.

A *simplified robot dynamics algorithm* has been developed to overcome the drawbacks of the purely kinematic simulation. During each time-step, all external forces and resulting torques are summed up at the total center of mass of the humanoid robot. Only for the center of mass dynamics calculations are performed. Relative motions of all other elements of the robot are calculated using direct kinematics. The basic version of the algorithm uses a center of mass and an inertia matrix of the robot which are in a constant position relative to the robot's base. For more realistic motions, the center of mass and the inertia tensor can be calculated for each time step based on masses assigned to the robot's bodies. Both versions allow for a rich variety of motions. The second version is especially useful for humanoid robot motions during which the robot's overall mass distribution significantly changes as for falling down, standing up or balancing. As these algorithms do not rely on too specific assumptions on the robot's kinematical structure, they are not limited to humanoid robots.

A special strength of `MuRoSimF` is, that *algorithms for full MBS forward dynamics* like composite rigid body or articulated body algorithms [3, 4] can be incorporated as well in case a higher level of detail in physical motion simulation is required for one or more of the robots. Also for numerical integration not only the common Euler's method but also higher order methods with variable step size may be employed.

3.3 Collision detection and handling

To simulate the interactions of a simulated robot with its environment which are caused by physical contacts the collisions between the robot's bodies and the environment must be detected and handled. Collision detection and handling are treated as separate modules for which different algorithms of different complexity or level of detail can be applied easily.

Collision detection is based on a constant property assigned to an object describing its shape as well as the object's position and orientation. Currently the primitive objects box, ellipsoid, cylinder and plane are supported. To avoid the $O(n^2)$ complexity of checking each object for collision with any other object, collision detection can be activated during setup of the simulation for each pair of objects individually. This process can be automated by defining sets of objects which do not need to be intersected with each other. For specialized simulations it is also possible, to use only selected bodies of a robot for collision detection (e.g., only the feet, if no other application than humanoid walking is considered). A collision is described by the penetration depth c_{depth} of the two bodies, the position $\mathbf{c}_{pos} \in \mathbb{R}^3$ of the contact point and the direction $\mathbf{c}_{normal} \in \mathbb{R}^3$ of the normal force pushing the bodies apart.

Collision handling is based on the collisions detected before. Currently a *soft* collision model is implemented, allowing colliding bodies to penetrate. Collisions are handled by calculating the resulting normal and frictional forces. To allow for different kinds of surfaces, additional parameters describing the contact situation are used. These parameters depend on the two colliding surfaces and are stored for each pair of surface types once. The normal force

$$\mathbf{f}_{normal} = \begin{cases} 1 \cdot s_c \cdot c_{depth} \cdot \mathbf{c}_{normal} & \text{if objects are getting closer,} \\ s_b \cdot s_c \cdot c_{depth} \cdot \mathbf{c}_{normal} & \text{if objects are separating.} \end{cases} \tag{1}$$

is calculated using a spring model with the spring constant s_c. The scaling factor $0 \leq s_b \leq 1$ is used to model different forces depending on the objects' relative velocity, allowing for different kinds of impact. Frictional forces are calculated using a viscous friction model depending on the relative velocity of the colliding bodies and the constant μ. As only one contact point is calculated for each pair of bodies, a pseudo-friction depending on the relative angular velocity is calculated which is used to stabilize standing bodies

$$\mathbf{f}_{friction} = \mathbf{v_{rel}} \cdot s_\mu \cdot f_{normal} \ \ , \ \ \mathbf{n}_{friction} = \omega_{\mathbf{rel}} \cdot s_\nu \cdot f_{normal}. \tag{2}$$

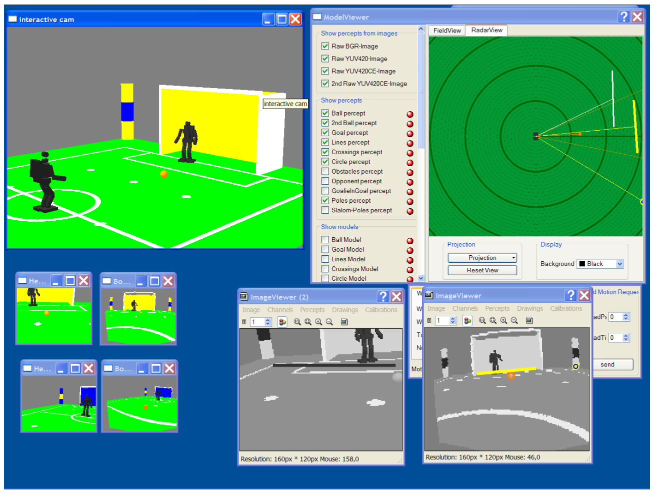
3.4 Visualization and camera simulation

Real time rendering for visualization of the simulated scene is based on OpenGL. It is possible to display any property of any simulated object: all objects which need to be displayed are registered with the rendering module which keeps a set of renderers for the properties of interest and matches any object having a specific property with the respective renderer. Complexity and realism of the rendering process can be adjusted easily by exchanging the respective renderers. Therefore, scene rendering can be adjusted to different needs and levels of detail.

Camera simulation is based on the visualization module. After rendering the scene from the camera's point of view, the created image can be post-processed. Distortion caused by a camera lens (cf. Fig. 2) is simulated by moving the pixels according to the camera model from the well known camera calibration toolbox for Matlab [14]. This approach enables easy modeling of real cameras by calibrating them with the freely available toolbox. By changing the level of detail (and therefore realism) of the rendering algorithms used, the camera simulation can be adjusted to different cases and purposes.

4 Results

Using `MuRoSimF` various simulators for specific applications, e.g., for teams of humanoid robots or mixed teams of humanoid and wheeled robots can be re-



Show percepts from images
- [✓] Raw BGR-Image
- [✓] Raw YUV420-Image
- [✓] Raw YUV420CE-Image
- [✓] 2nd Raw YUV420CE-Image

Show percepts
- [✓] Ball percept
- [✓] 2nd Ball percept
- [✓] Goal percept
- [✓] Lines percept
- [✓] Crossings percept
- [✓] Circle percept
- [ ] Obstacles percept
- [ ] Opponent percept
- [ ] GoalieInGoal percept
- [✓] Poles percept
- [ ] Slalom-Poles percept

Show models
- [ ] Ball Model
- [ ] Goal Model
- [ ] Lines Model
- [ ] Crossings Model
- [ ] Circle Model

Background: Black

Resolution: 160px * 120px Mouse: 158,0

Resolution: 160px * 120px Mouse: 46,0

Fig. 2. Left: Simulation of motion and sensing of two humanoid robots, each equipped with an articulated camera in the head and a wide angle camera attached to the chest whose images are displayed below. Right: GUI of control application displaying the striker robot's percepts in robot centric coordinates and within the camera images.

alized. For each robot the simulation algorithms for motion, sensing, collision detection and handling can be chosen individually. Thus a high scalability of the level of detail can be achieved for tailoring the real-time simulation accuracy and computational complexity to the current needs.

The simulator's performance has been measured for scenarios from the RoboCup humanoid league using a standard laptop computer (IntelCentrinoDuo CPU (1.66 GHz), 1GB of RAM, Intel 945GM graphics chip set) with the simulation running single-threaded. Each humanoid robot motion model consists of 21 rotational joints [2] and the robot is equipped with two camera systems (cf. Fig. 2). The performance measurements are summarized in the following table:

Real-time simulation of	*frame rate*	*robots*
Kinematic motion	100	10
Simplified dynamics motion	1000	8
Kinematic motion and one camera	100 resp. 20	6
Dynamic motion and one camera	1000 resp. 20	5
Kinematic motion and one camera with distortion	100 resp. 20	3
Dynamic motion and one camera with distortion	1000 resp. 20	3

The simulator has been used successfully for testing the software modules for vision, behavior control and self localization of robots in different scenarios from the RoboCup Humanoid League (Fig. 3). Further images and videos can be obtained from `www.dribblers.de/murosimf`.

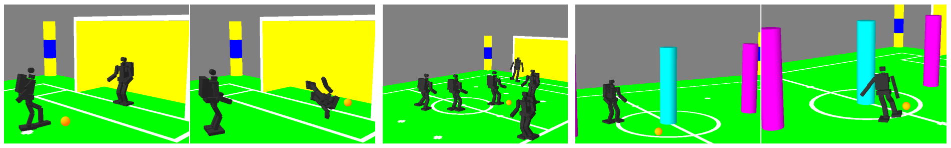


Fig. 3. Left: Penalty kick. Middle: 3 versus 3 soccer game. Right: Slalom challenge.

5 Conclusions

`MuRoSimF`, a new simulator framework has been presented which enables real-time simulation of motion, sensing and interaction with the environment for SIL-testing of onboard control software modules for teams of humanoid robots. It supports a scalable level of detail and a flexible exchange of algorithms for different simulation subtasks for different robots in the same multi-robot simulation. Results have been presented for scenarios of the RoboCup humanoid league including slalom challenge, penalty kick and 3 versus 3 soccer games.

Acknowledgement. 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, and O. von Stryk. Modular software architecture for teams of cooperating, heterogeneous robots. In *Proc. IEEE Intl. Conf. on Robotics & Biomimetics*, pages 613–618, Kunming, China, Dec. 17-20 2006.
2. M. Friedmann, J. Kiener, S. Petters, H. Sakamoto, D. Thomas, and O. von Stryk. Versatile, high-quality motions and behavior control of humanoid soccer robots. In *Proc. Workshop on Humanoid Soccer Robots of the 2006 IEEE-RAS Int. Conf. on Humanoid Robots*, pages 9–16, Genoa, Italy, Dec. 4-6 2006.
3. R. Featherstone and D. Orin. Robot dynamics: Equations and algorithms. In *Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 826–834, San Francisco, CA, USA, April 2000.
4. M. Hardt and O. von Stryk. The role of motion dynamics in the design, control and stability of bipedal and quadrupedal robots. In G.A. Kaminka, P.U. Lima, and R. Rojas, editors, *RoboCup 2002 Intl. Symposium (Robot Soccer World Cup VI)*, volume 2752 of *Lecture Notes in AI*, pages 206–223. Springer-Verlag, 2003.
5. R. Smith. ODE - Open Dynamics Engine, www.ode.org. 2007.
6. T. Laue, K. Spiess, and T. Röfer. SimRobot - a general physical robot simulator and its application in RoboCup. In A. Bredenfeld et al., editor, *RoboCup 2005: Robot Soccer World Cup IX*, number 4020 in Lecture Notes in AI, pages 173–183. Springer, 2005.
7. F. Otsuka, H. Fujii, and K. Yoshida. Development of 3D dynamics simulator with omnidirectional vision model. In *RoboCup 2006: Robot Soccer World Cup X*, 2006.
8. J. C. Zagal and J. Ruiz-del Solar. UCHILSIM: A dynamically and visually realistic simulator for the robocup four legged league. In *RoboCup 2004: Robot Soccer World Cup VIII*, pages 34–45, 2004.
9. S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. USARSim: a robot simulator for research and education. In *Proc. of the 2007 IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2007.
10. M. Zaratti, M. Fratarcangeli, and L. Iocchi. A 3D simulator of multiple legged robots based on USARSim. In *RoboCup 2006: Robot Soccer World Cup X*, 2006.
11. N. Koenig and A. Howard. Gazebo - 3D multiple robot simulator with dynamics, website http://playerstage.sourceforge.net/gazebo/gazebo.html. 2003.
12. Player/stage project website, http://playerstage.sourceforge.net/. 2006.
13. Cyberbotics Ltd. Cyberbotics Webots, www.cyberbotics.com/products/webots/.
14. J. Bouguet. Camera calibration toolbox for Matlab, www.vision.caltech.edu/bouguetj/calib_doc/.