

# Robot simulation, collisions and contacts

Joshua G. Hale, Benjamin Hohl, Eduardo Martin Moraud

**Abstract**—A software platform is essential for development, experimentation and research involving humanoid robots. Such a platform should naturally provide a software interface for controlling the robot, and make the sensing data gathered by the robot in real-time available to the task programmer in an organized manner. This paper presents our work focused on developing an efficient and accurate simulator which has been applied to three humanoid robots: I-1, CB-i and Hoap-2. We present the structure of the control and simulation software, the handling of ground contact using a constraint manipulation method that incorporates Coulomb friction exactly, an optimization which trades accuracy for efficiency, and we describe the detection and avoidance of auto-collisions among the limbs of a robot itself.

## I. INTRODUCTION

A software platform is essential for development, experimentation and research involving humanoid robots. Such a platform should naturally provide a software interface for controlling the robot, and make the sensing data gathered by the robot in real-time available to the task programmer in an organized manner. Ideally, sensor data should be further exploited to provide higher level data such as link coordinate frames, Jacobians and so on, which may be useful for a range of tasks and aid in controller development. Users of humanoid robots also benefit from having an on-line interface by which to command the robot, activate tasks manually, gather, log and analyse data. The third major component of a robot platform is simulation. A humanoid robotics platform is therefore a software project of considerable scale and it makes sense to provide its functions in a consistent manner so as to minimize the effort of learning how to use it, and maximize the re-usability of control code in a hierarchical manner and with different robots. The platform we have developed embodies these ideals, with a consistent interface to a number of different humanoid robots at the level of the software interface and the on-line interface, and by offering accurate simulation with empirically calibrated and experimentally validated modeling of contact and friction.

Simulation is a fundamental tool in humanoid robotics, fulfilling many important roles. For example, it is used to validate controllers with respect to safety and performance considerations; the controller development cycle of

This work was supported by the JST-ICORP Computational Brain Project. Joshua G. Hale is with the JST-ICORP Computational Brain Project, 4-1-8 Honcho Kawaguchi, Saitama, Japan, and the ATR HRCN Group, 2-2-2 Hikaridai Seika-cho, Soraku-gun, Kyoto 619-0288, Japan. [jhale@atr.jp](mailto:jhale@atr.jp)

Benjamin Hohl is with the Institut für Technische Informatik (ITEC), Universität Karlsruhe, Germany, and the ATR HRCN Group. [hohl@gmx.de](mailto:hohl@gmx.de)

Eduardo Martin Moraud is with the INRIA Nancy Unit - Campus Scientifique - BP 239 - 54506 Vandoeuvre-lès-Nancy Cedex, France, and the ATR HRCN Group. [martined@loria.fr](mailto:martined@loria.fr)

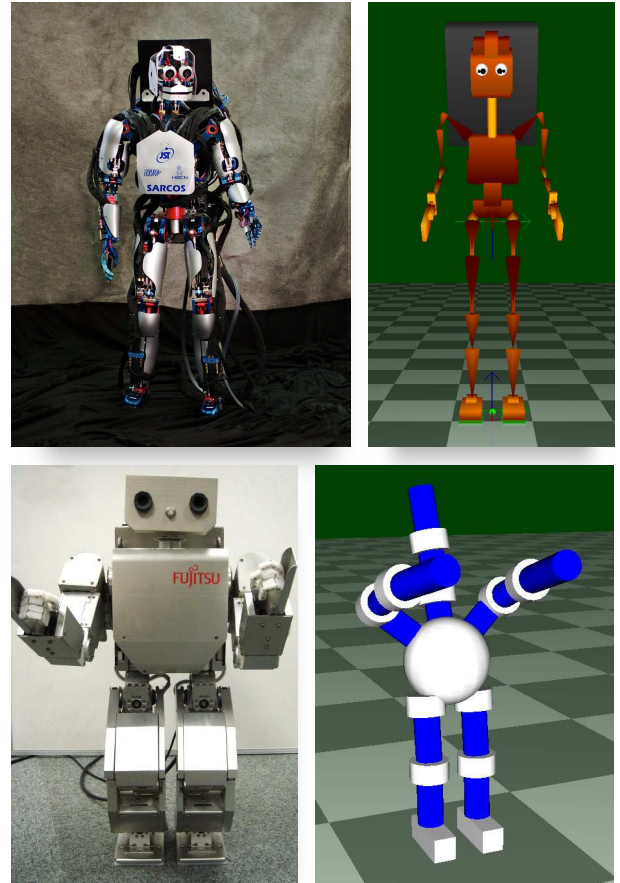


Fig. 1. The humanoid robots CB-i and Hoap-2, and their simulated counterparts

programming, testing and improvement is made more convenient through simulation, which should be easier, safer and quicker to execute than experiments on a real robot; the repeatability of environmental interactions and sensor data makes it easier to debug control code; multiple simulations may be performed automatically to gather feedback data for processes such as reinforcement learning; simulation facilitates parallel development by multiple researchers who must share time working with a single robot; simulation also allows hardware modifications to be tested before making physical modifications.

In order to properly support these applications and obtain meaningful results, it is crucial to maintain highly realistic simulation. Since ground contact and friction can have a dramatic influence on the quality and repeatability of mo-

tion for free-standing humanoid robots and are generally handled in an imprecise manner, we focused attention on this area. We have thus developed a highly precise simulator with an efficient method for resolving ground contact. The parameters governing ground contact have been identified experimentally, and the accuracy of the method has been validated empirically. The key goal of our simulation environment is thus high fidelity emulation of actual robot behavior. This encompasses not only the physical responses of the robot but also the information exchange of control and sensing data. When the simulation and information pathways are sufficiently realistic that the control software used to govern robot behavior cannot discriminate between a real and simulated robot, the interface may be described as ‘transparent’.

One of the benefits of engineering such transparency is that control code, and control components can be developed and validated confidently in simulation. By ‘control components’ we refer to the many re-usable functions that may be adopted by controllers intended for different specific tasks. For example, we have developed (i) automated balancing algorithms applicable with position-based[1] and force-based[2], [3] actuation, (ii) auto-collision detection, prevention and avoidance, and (iii) a real-time motion capture interface. In order to maximize the utility of such control components it is desirable to prepare the environments for different robots in as consistent a manner as possible so as to ensure portability with little or no effort.

Our simulation environment embodies this characteristic of generality. We have adapted it for use with three humanoid robots I-1, CB-i, and Hoap-2. CB-i is a largely identical, somewhat improved version of I-1 and we therefore discuss only CB-i and Hoap-2 in this paper. Hoap-2 is a small sized (50cm) humanoid robot manufactured by Fujitsu [4]. It has 25 degrees of freedom:  $2 \times 6$  DOFs legs;  $2 \times 4$  DOFs arms;  $1 \times 2$  DOFs neck;  $1 \times 1$  DOF neck;  $2 \times 1$  DOFs hands. It is actuated by electronic servo motors and is position-controlled. CB-i [5] is a human-sized humanoid robot manufactured by SARCOS, and has 50 degrees of freedom:  $2 \times 7$  DOFs arms;  $2 \times 7$  DOFs legs;  $2 \times 2$  DOFs eyes;  $1 \times 3$  DOFs neck;  $1 \times 3$  DOFs torso;  $2 \times 6$  DOFs hands. The arms, legs, neck and torso are actuated hydraulically, and active compliance is made possible with joint position and force sensors installed at each of these joints. These joints may be controlled at 5kHz in any of three modes: position, velocity or force. The eyes are driven by electric servo motors, and the fingers on the hands are pneumatic and passively compliant. The speed and range of joint motion have been made to closely match that of an average person [6]. Both robots may be seen, along with their simulated counterparts in Fig. 1.

This paper is organized as follows. Section II presents the design of the robot control and simulation software, including its user interface, task server, and simulation components. Contact handling is discussed in Section III, and the detection, prediction and avoidance of auto-collisions is described in Section IV.

## II. SYSTEM STRUCTURE

A structural overview of the simulation environment is shown in Fig. 2. It is multi-threaded, so that four processes can be executed in parallel: *simulation*, *task server*, *console input*, and a *GUI*. The data path for these processes is via shared memory, which is protected using mutexes to prevent access conflicts. The shared memory contains simulation and task times, so that these processes may be synchronized when the software is used in simulation mode. In addition to the shared memory data path, the *input* process may execute functions provided by the *task server* thus affecting task management. These functions however, execute in the *input* thread.

The data-path via shared memory relays control and sensing information. Control data includes, for each joint: the desired position, velocity and actuator force, as well as the gain parameters for the position, velocity and force commands, and a feed-forward force command. Sensing data includes for each joint: the position, velocity and load, as well as the orientation of the base link, and external forces acting on the soles of the feet.

A transparent interface to the simulator or to the robot itself is provided via the *simulation* thread. When a task controls an actual robot, the *simulation* thread executes, but the command information is transmitted by UDP to another host, and sensing information is likewise received. The remote host may be a motor server executing on the robot itself, or it may be another simulator executing a motor server emulation task. This emulation task simply replicates the motor server UDP communication process, passing sensing data out and control data in, as a task to be executed by a local simulation process. A task may thus execute without knowledge of whether it is executing on a local simulation process, or via UDP, and in the latter case whether the remote process is a simulation, or the actual robot. This yields a transparent interface that, besides controlling the robot, may be used to validate possible network issues, and also split the burden of simulation and task execution in parallel over a network.

Regarding timing, UDP communications are time stamped, and the clock at the *motor server* or remote simulation is provided to the *task server*. The motor server communication must time-out if control information is not available, and in this case replicates the most recently received control data. The controller communication however, need not time-out, and may wait indefinitely for sensing data (which may arrive time-stamped faster, or slower than real-time when simulation is used). In actuality a delay of 3 seconds (at either end) is treated as an indication that the network connection has failed, and remote operation is aborted.

Functionality common to both simulation and task execution is provided in the simulation process. The *simulation* thread augments the sensing data in shared memory with various useful structures including the coordinate frames of all the links, the centre of mass Jacobian in both the world

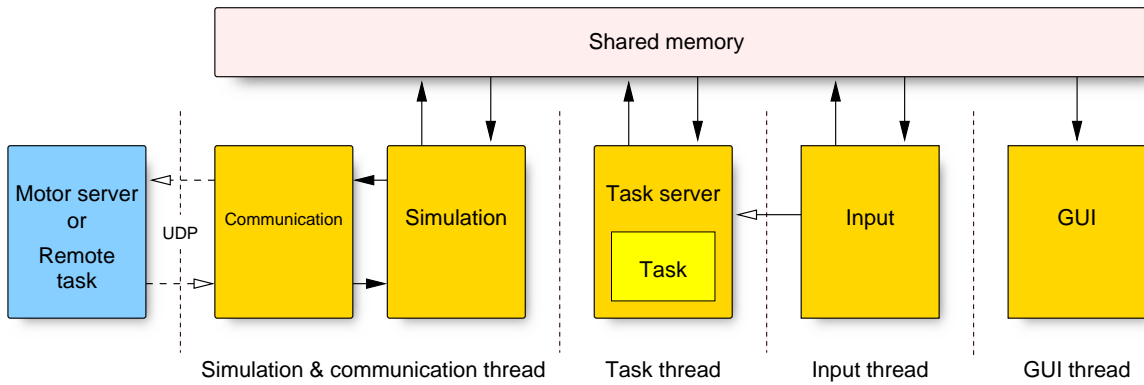


Fig. 2. System overview

and base frames, position and orientation Jacobians of the contact points on the feet, the foot coordinate frames and hand coordinate frames, and absolute position of the base link. The latter is computed using forward kinematics by combining base orientation and joint angle sensor data with assumptions about which feet are static, or by integrating information from gyro sensors and force plates. Contact state can be determined automatically using the foot contact force sensor, or be specified manually within task code according to arbitrary principles. The augmented data may be computed by inserting sensing data from the motor server of the actual robot, or extracted directly from the state of the simulation. The *task server* then provides a ubiquitous programming environment for task development augmented with the information calculated in the simulation thread. In case execution time is critical, a task may activate or deactivate the computation of particular augmented data by setting flags stored in shared memory. Bearing in mind robot-specific considerations such as joint names, the number of DOFs and availability of position-command or force-command actuation, task control code may be applied seamlessly to either robot.

#### A. User interface

The *GUI thread* shown in Fig. 2 acts independently as an ‘observer’, and makes no adjustment to shared memory. GUI controls affect only viewing parameters, rendering style and what data to display. Rendering requests may be made in task code running in the *task server* thread in order to display 3D cursors such as estimated ZMP coordinates. The *GUI thread* may be terminated, if for example, multiple batch simulations are required to gather data for controller trial and improvement.

The simulation state is displayed in the *GUI thread* using OpenGL, and the GUI itself was designed using Qt3 (see Fig. 3). The GUI provides visualization of contact forces, contact states (see below), CoM, ZMP *etc.*, as well as user defined points of interest specified in task code. In addition, the environment may be viewed from the robot’s perspective in order to assess its field of view for visual processing tasks. When the software is used to control an actual robot, the sensor data from the robot, including any additional gyro

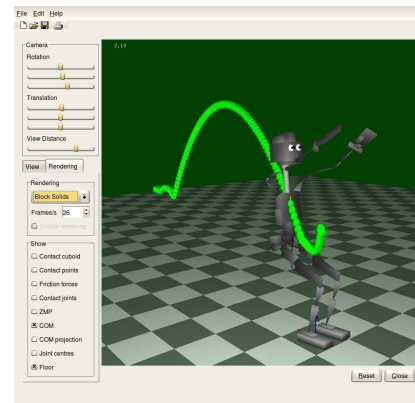


Fig. 3. A screen shot of the simulator showing the Qt-based GUI, and visualization functions applied to back flipping motion recorded in simulation (the green line is the trajectory of the CoM)

sensors, force plate readings, *etc.*, are combined to deduce the absolute state of the robot which is displayed in the same manner as a simulated robot.

Besides the rendering control, and visualization selectors (COM, ground reaction forces, ground contact state, *etc.*) present in the GUI, the environment provides a console-based interface. Tasks may be configured and initiated; simulated or actual data may be recorded, saved, loaded, played back and analyzed at different rates and using sophisticated cueing commands; recorded data may be output in configurable text formats for easy import into graphing applications; environmental parameters may be set, such as floor friction levels, joint frictions, arbitrary external forces and so on; and various parameters of the robot under simulation may be reconfigured, such as the mass properties of the backpack carried by CB-i.

#### B. Task server

The function of the task server is to provide a consistent software interface for robot control, and facilitate the selection and execution of control tasks on-line. It provides numerous tools for both on-line and software-level state analysis and control.

The software level interface requires the control programmer to specify two functions and register them with the task server at compile time. These are the task's *run* function and *initialization* function. A *change* function may also be optionally specified. The task execution process is invoked via the input thread, which identifies the appropriate task and immediately executes the appropriate *initialization* function. Since this function executes in the input thread it may for example interact with the user to obtain parameter values. It is not obliged to execute in real-time. Besides setting parameters, it typically used to allocate memory for the run function. Once the initialization function terminates, the task server is informed that the task is active. From this point on the run function is called at the frequency of the task server (0.5-1kHz) while the task is active. The run function is executed in the task server thread, and is expected to respect real-time requirements. If the programmer includes a *change* function, then this may be called from the input thread (where it will execute), and thus the user may adjust the parameters of a given task while it is executing. The return value of these functions is interpreted by the task server as an indicator as to whether the task should continue executing, or has completed.

The task server prepares various data so that at the task code level, global variables are made available containing the following information: current state of the robot's joints; desired state of the joints; distribution of ground-contact forces; active contact points; polygon of support<sup>1</sup>; point of optimal stability<sup>2</sup>; base position and orientation; link transforms; contact point Jacobians; centre of mass Jacobian [8]; end-effector Jacobians; and joint actuator gains.

In addition to this task-level software interface, the task server also provides various functions that the user may invoke on-line via the input thread. These include: instantaneous state storage (and restoral, in the case of simulation), motion recording and storage, state queries (including data that may be reported directly such as the actual sensor readings for the joint angles, *etc.*, and computed data, *e.g.*, oil flow), direct manipulation of the robot's joints via a posture control task, and motion playback via a posture interpolation task that fits smooth derivative minimized splines [9] through knot points defined by a series of postures.

### C. Simulation

The simulation engine was developed in C++. Efficient forward dynamics equations for the Hoap-2 and CB-i models were generated as C functions using SD Fast, and symbolically optimized prior to computation. These functions are integrated using a Runge-Kutta adaptive step-size method. In the case of CB-i, the simulator models the 34 hydraulically actuated joints of CB-i dynamically, but models the 4 DOFs

<sup>1</sup>The polygon of support is the convex hull of the active contact points, and may be calculated in various ways. See [7] for details.

<sup>2</sup>We defined the point of optimal stability to be the mid-point of the locus of centres of discs of maximal size completely contained within the convex hull. For simple convex polygons, this point may be computed by finding the largest disc tangential to each edge pair, and testing these discs for containment in the hull.

of the eyes kinematically. The eyes are sufficiently light, and powerfully actuated that for the purposes of modelling it is reasonable to ignore their inertial interactions. Incorporating them into the dynamic model on the other hand, necessitates a very fine time-step because being so light -subject to very stiff control, and at the end of the kinematic chain, they are highly sensitive to the motion of the body.

In addition to the task controller framework for controlling the internal forces generated by robot actuators, the simulator provides a software interface for defining external forces. This may be used to model interactions with the external environment in arbitrary ways. Force models are defined analogously to task code, by registering an initialization and run function. The former is executed once, to initialize data and obtain parameters via the *input thread*. The run function is executed according to the requirements of the adaptive step-size integrator. It is provided with the current state and time and must compute the appropriate external forces. Arbitrarily complex patterns of external force may thus be incorporated into simulation, modelling for example, joint frictions, physical interactions with a human, and manipulations involving external objects with arbitrary mass and inertia characteristics.

The key principle of the simulator is its highly precise emulation of actual robot behavior in the real world. Having established an accurate model of the intrinsic dynamic behavior of a robot, the main source of divergence from reality is the interaction between the robot and it's external environment. In particular, a precise model of ground contact and friction is necessary in order to ensure the fidelity of simulated motion. An accurate and empirically-calibrated contact and friction-force resolution method was therefore developed, and is described in the following section.

## III. CONTACT HANDLING

The simulator provides an efficient contact handling method that applies Coulomb friction exactly [10]. The importance of handling contact and friction forces accurately and efficiently is well known [11], and many methods have been proposed using for example, impulse methods [12], [13], [14], analytical methods [15], [16], [17], springs [18], [19], [20] and pseudoinverses [21]. Impulse methods tend to be iterative, progressively refining a solution by propagating energy between bodies in a series of collisions handled with instantaneous impulses. As such, the results are oriented towards visual plausibility rather than realism. Many of the analytical methods are based on the complementarity principle [15], which although not always consistent with reality [22] may usually be solved with linear complementarity (LCP) methods typically based on Lemke's algorithm. Solution is not guaranteed however, and a strict interpretation of Coulomb friction in fact demands a non-linear complementarity problem which is usually avoided by approximating the friction cone using a polyhedral friction pyramid [16], [12], [14], [17]. Penalty-based methods such as spring-damper systems are trivial to implement, but must be stiff, necessitating small time-steps during iteration, and do

not adequately account for friction, although some progress has been made in this direction [20]. The Moore-Penrose pseudoinverse method [21] also does not handle friction adequately, produces an approximate solution and is slow to compute because the most applicable techniques for updating the pseudoinverse are inadequate for this problem [23]. The method presented in [24] uses the pseudoinverse to solve for an infinitesimal screw minimizing the penetration depth of bodies at representative contact points, by formulating a linear program for the screw using a non-linear objective function. The actual contact forces in their method are resolved as spring-damper forces thus compounding the computational burden. The tangential (friction) forces are also projected onto a polyhedral friction cone as a final step in order to model static and dynamic friction appropriately, although such polyhedral approximations yield anisotropic responses to friction.

In contrast, the method we implemented avoids the need for integrating stiff spring-dampers by utilizing constraints which can be realized by means of, for example, Lagrange multipliers, and does not require time-consuming pseudoinverse computations. The friction cone is handled in an isotropic fashion so that contact is modelled both efficiently and accurately.

#### A. Exact constraint-based method

The method we developed models contact between the humanoid's feet and a planar ground surface, and explicitly identifies the contact state of each foot. For a rectangular footed biped, there are 361 possible contact states. Each foot may be contacting in one of 19 ways: at one of four vertices, one of four edges, with its face, or not in contact at all, and the contact may be static or dynamic (see Fig. 4). Static friction is applied by incorporating the appropriate motion constraints into forward dynamics computations using Lagrange multipliers, and dynamic frictions are integrated as additional forces. The paradigm of contact state enumeration has been discussed previously [25], [26] but the criteria for contact state selection in preceding work are different, and state analysis does not guarantee to reveal the appropriate contact state.

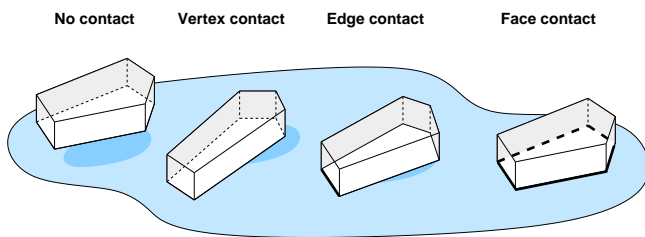


Fig. 4. Contact states

The key component of the method is the identification of the appropriate contact state. This computation is interleaved with forward dynamics integration steps, and makes use of coherence between subsequent time-steps to reduce the burden of model selection -by partitioning the work of constraint maintenance and constraint selection, the component

that often remains unchanged over subsequent time-steps (constraint selection) may be optimized. An efficient priority queue is used to rank contact states and select the appropriate state at any given instant, as may be seen in Fig. 5.

Contact states are evaluated according to the following criteria: *geometric validity* (ensuring that the foot does not penetrate the ground plane, and that vertices of the foot are not constrained unless they lay on the ground plane), *kinematic validity* (ensuring that vertices of the foot do not move or accelerate through the ground plane, and that vertices that are naturally separating from the ground are not constrained) and *minimal constraint* (ensuring that the minimum of possible constraints are applied to ensure valid motion).

It is sufficient to facilitate the binary comparison of a pair of contact states, in order to identify the correct state according to the process indicated in Fig. 5. The comparison procedure is summarized in Fig. 6. The procedure accounts for manually specified 'forced states' (mainly used for testing and controller development) and considers each of the corner points of the feet according to whether or not they are 'clamped', *i.e.*, constrained to remain on the ground plane. It involves small threshold values preventing chattering: distance tolerance  $d$ , velocity tolerance  $v_t$  and acceleration tolerance  $a_t$ , which were set to 1mm,  $0.1\text{ms}^{-1}$  and  $0.1\text{ms}^{-2}$  respectively in our model.

#### B. One-step constraint-based method

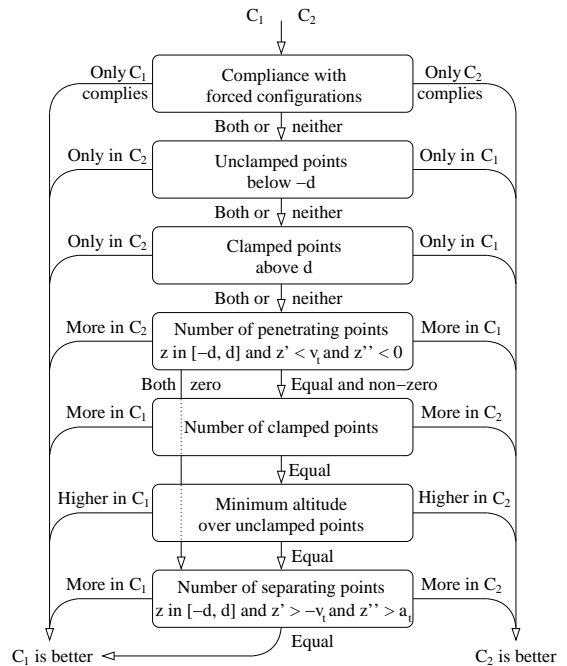


Fig. 6. Procedure for comparing candidate contact states

In addition to the method described above which identifies the best contact state in every time-step, we developed a faster one-step method which trades-off accuracy for computation time. There are two factors which make it difficult to identify the correct contact state in a single step without

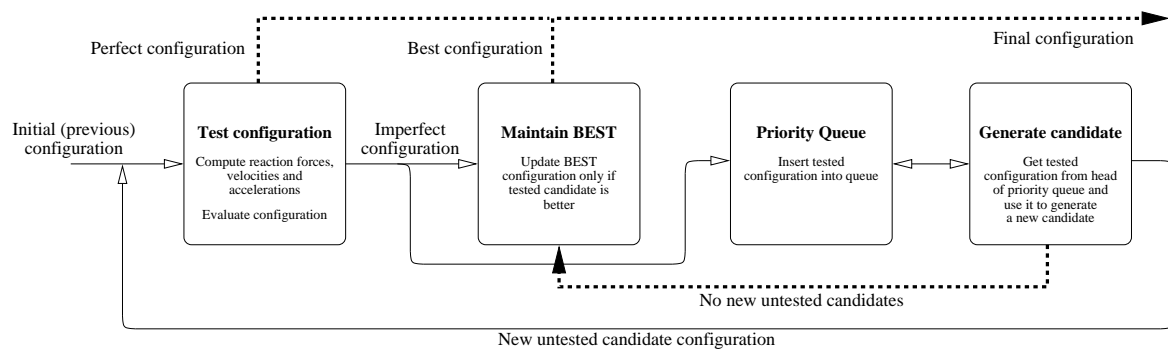


Fig. 5. Control flow

testing a number of states (testing involves setting the constraints for a given state, and calculating the resultant contact forces and accelerations to discover if they are valid). Firstly, the contacts are handled as interactions between a number of discrete contact points and the ground, and changing the forces acting on the feet may affect the accelerations of each point in ways that are hard to predict. Secondly, numerical computation on computers is not exact and it is necessary to incorporate a degree of tolerance into the selection of the appropriate state so as to prevent chattering between contact states, and handle cases where inconsistent states arise (such as slight penetrations of the floor of an order equal to the numerical inaccuracy of floating point computations) and in effect soften the infinitely hard rigid body model insofar as is necessary to achieve a stable simulation of the real world. This tolerance implies that more than one state may be acceptable. It is therefore necessary to find the best state in each of these two cases -no perfect solution, or multiple acceptable solutions.

The one-step method adopts a tolerant approach to state selection, by aiming to select the state in a single step. Only the current properties of the model and the information regarding the current contact state are utilized. Each contact point is considered independently, and the decision regarding whether it should be clamped is taken in a manner analogous to the hierarchy expressed in Fig. 6, *i.e.*, *geometric* (position-related) factors are prioritized, followed by *forces* (avoidance of negative, and thus attractive, support forces), followed by *kinematic* factors (unclamping of points near the floor with a velocity away from the floor, and *vice versa*). This reveals whether each point should be clamped or unclamped when considered independently. Since the evaluation is based purely on the current state of the contact points under the constraints imposed by the current contact state, besides their positions, it is necessary to refer to both the velocities of, and forces acting on the contact points in order to predict the tendency of each point to rise from, or penetrate the floor. Furthermore, since the points are considered independently there is no guarantee that the selection of points to clamp may be physically consistent with one of the four clamping patterns (face, edge, vertex or free). A weighted sum is thus calculated for all of the relevant possible clamping

patterns which includes an error term for each vertex that is inconsistent with the ideal clamping pattern. The pattern with the lowest cost is selected. The weights are set according to the hierarchy of selection criteria and the total number of contact points in order to ensure a strict selection hierarchy.

#### IV. AUTO-COLLISION

The ability to deal with situations in which a robot risks striking its own body is beneficial for two reasons. Humanoid robots are generally expensive and sensitive devices, so repairs can be expensive and time consuming. Violent auto-collisions can be dangerous to a robot's environment which may include humans observing or interacting with the robot. The primary concern is thus the safety of humans, as well as the robot and its environment. Secondly, auto-collisions may interfere with a robot's ability to accomplish a task. It is therefore desirable to resolve potential auto-collisions in order to maintain smooth and effective functioning. The cases that must be handled may be split into two types according to whether the intended motion is known in advance, or is the result of some unpredictable real-time process. An effective solution to the latter case subsumes the former case, and we have thus adopted such a general method. This does however disregard the potential for more task-oriented solutions which take account of a complete motion sequence, and might accommodate task-specific goals. Real-time motion capture and performance by a humanoid robot is an example of the latter case, since it is a task for which the future target state is not known in advance.

The functions of an auto-collision handling system may be summarized as follows: *collision detection*, *collision prediction* and *collision avoidance*. The requirements of the system are: close adherence to desired motion, few assumptions about future trajectory, real-time performance, and collision free output.

We implemented a hierarchical modular solution by dividing the work into four components designated *spatial*, *kinematics*, *detection* and *avoidance*.

The *spatial* component deals with the geometry of the robot model, loading and parsing its elements and essentially initializing the *kinematics* and *detection* components. We used the Qhull software to prepare the robot model as an articulated collection of convex polyhedral elements. The

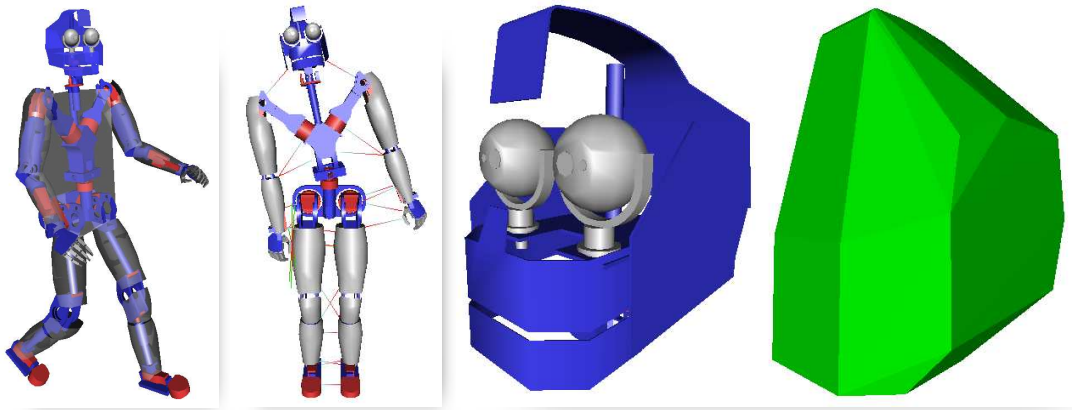


Fig. 7. Original VRML model of CB-i, image augmented with the minimum distance vectors between each body indicating a contact between the right hand and thigh, and an example convex hull generated for the head.

specific details of the model used for CB-i are discussed in Section IV-A. The original CB-i model is shown in Fig. 7, which includes an image augmented with the minimum distances between each body calculated using the auto-collision system which indicates a contact between the right hand and thigh, and an example convex hull enclosing the head.

The *kinematics* component provides the functions needed to calculate the absolute positions of the elements of the model given the joint angles of the robot, as well as integrating joint velocities and accelerations in order to extrapolate future states of the robot. The maximum possible joint velocities and joint ranges according to the physical capabilities of the robot were used to cap the results of integrating the joint accelerations to within reasonable bounds so as to avoid overly conservative collision predictions. Future states of the robot were thus predicted by extrapolation based on the second derivative, and submitted to the detection component.

The *detection* component was used to analyze a given state of the robot and identify the set of all colliding links. The SWIFT++ library was used to perform the collision tests. A two-layer hierarchy of tests was adopted, using axis-aligned bounding boxes with simple sweep tests to quickly eliminate distant bodies as a first step, followed by the Lin-Canny feature tracking algorithm based on Voronoi regions to identify the shortest distance between potentially penetrating bodies. The latter algorithm takes advantage of coherency among successive collision tests applied to the same set of bodies. The set bodies was preprocessed by hand to eliminate links which are incapable of auto-contact. This includes distant bodies such as the foot and the torso which could not contact due to the kinematic configuration of the robot, as well as adjacent links whose physical joint limits prevented mutual contact.

Several approaches were considered for the *avoidance* component, including potential fields, path planning, freezing and random perturbations. Potential fields acting as forces repelling limbs or penalizing motion generally require time consuming dynamic simulation, and cases of ‘near collisions’

that it is desirable to permit are not handled well. Path-planning methods are well-suited to the offline case when data regarding a whole target motion, possibly augmented with particular goals is available. Instead we adopted simple kinematic constraints limiting the joint angles, speeds, accelerations and jerks based on linear interpolations in joint space. Only reachable cases were considered, thus avoiding the need to handle special cases of temporarily adjusted target postures such as avoiding target motion that involves auto-contact, or catching-up with an offset but collision free target motion.

Four methods were implemented, trading computational speed for solution quality to different extents. The simplest, trivial solution (Method 1) is to freeze the robot completely when a potential collision is imminent or when the target posture involves a collision. While this solution is not at all sophisticated it may be advantageous in certain situations, such as when a human operator is controlling the robot in real-time and can respond intelligently to the difficulties the robot may experience. The second solution (Method 2) refines this method by freezing only the joints which are involved in the collision. This provides more flexible performance but requires a larger number of collision tests in order to determine which joints may be considered mobile. Method 3 attempts to automatically resolve the discrepancy between the target motion and permissible collision free motion. Random perturbations of the affected joints (those frozen by Method 2) are used in an attempt to find postures which move individual joints closer to their target positions. Lastly, Method 4 refines this process again by using a heuristic algorithm to move the affected joints away from the joint coordinates causing the collision prior to converging on the target posture as in Method 3. A detailed exposition an analysis of these methods is omitted for the sake of brevity, and they are summarized in Tab. I.

#### A. Evaluation

The following results evaluate the collision system’s computational performance in the case of the humanoid robot

Method	Characteristics	Avg. tests per cycle	% of CPU time at 10Hz
Method 1	Freeze all joints when a target posture is colliding	3	0.25%
Method 2	Freeze only joints involved when a target posture is colliding	3-6	0.5%
Method 3	Freeze affected joints and attempt random perturbations in these joints to find collisions free motion	15-30	2.5%
Method 4	Freeze affected joints and use a heuristic method to generate an intermediate posture prior to attempting random perturbations to find collision free motion	30-60	5%

TABLE I  
COLLISION AVOIDANCE AND RESOLUTION METHODS

CB-i. A VRML CAD model of the robot with 10,316 polygons was reduced to 3613 triangles representing the convex hulls of all of the bodies in the model using the Qhull software. The model was further reduced to 2627 triangles by eliminating irrelevant bodies such as the eyes which are entirely contained within the convex hull of the head. The resulting 19 bodies had between 232 and 6 triangles. Testing a single configuration of the robot required an average of  $114\mu s$  for coherent postures, and about twice this for unrelated postures, *i.e.*, around 4300-8700 tests per second. Method 1 required around 3 tests per cycle, Method 2 required a few more depending on the nature of the target motion, Method 3 required around 15-30 tests per cycle, and Method 4 required around 30-60 tests per cycle. For comparison, it was desired to execute the CB-i task-server at either 500Hz or 1kHz. A generous estimate for the expected computation time of Method 4 is thus around 5s per real-time second if it is applied to every cycle when the task server operates at 1kHz. However, it is not necessary to execute the collision system at such an intensive rate, and little benefit is thus gained. In fact it is quite reasonable to execute the algorithm at around 10Hz which would require around 5% of the processing in real time. All of the models are thus applicable in real time and guarantee collision free motion. With regard to the ability of models 3 and 4 to continue performing obstructed motions, many types of motion are handled nicely, but variations in the motion goals (*e.g.*, communicative gestures, exact positioning tasks, precise actions, *etc.*) mean that there are cases in which one of the models may be preferred.

## REFERENCES

- [1] Takamitsu Matsubara, Jun Morimoto, Jun Nakanishi, Sang-Ho Hyon, Joshua G. Hale, and Gordon Cheng, "Learning to acquire whole-body humanoid CoM movements to achieve dynamic tasks", in *IEEE Int. Conf. on Robotics and Automation*, Rome, Italy, April 2007.
- [2] Sang-Ho Hyon and Gordon Cheng, "Passivity-based full-body force control for humanoids and application to dynamic balancing and locomotion", in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, Sept. 2006.
- [3] Sang-Ho Hyon and Gordon Cheng, "Gravity compensation and full-body force interaction for humanoid robots", in *Proc. of IEEE-RAS International Conference on Humanoid Robots*, Genova, Italy, 2006.

- [4] Koichi Kimura, Shigeo Tateno, Hiyoshi Ishikawa, and Keita Ogino, "Development of the compact humanoid robot HOAP-2", in *Proceedings of the 21st Annual Conference of the Robotics Society of Japan*, September 2003, 1G29, (in Japanese).
- [5] Gordon Cheng, Sang-Ho Hyon, Jun Morimoto, Aleš Ude, Joshua G. Hale, Glenn Colvin, Wayco Scroggin, and Stephen C. Jacobsen, "CB: A Humanoid Research Platform for Exploring NeuroScience", *Advanced Robotics*, vol. 21, no. 10, pp. 1097–1114, 2007.
- [6] Alvin R. Tilley, *The Measure of Man and Woman: Human Factors in Design*, John Wiley & Sons, Inc., Revised edition, 2002, ISBN: 0-471-09955-4.
- [7] Joseph O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1994, ISBN 0-521-440343.
- [8] Tomomichi Sugihara and Yoshihiko Nakamura, "Whole-body cooperative balancing of humanoid robot using cog jacobian", in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2002, vol. 3, pp. 2575–2580.
- [9] Joshua G. Hale, *Biomimetic motion synthesis for synthetic humanoids*, PhD thesis, University of Glasgow, 2003.
- [10] Joshua G. Hale, "Contact handling with static and dynamic friction for dynamically simulated articulated figures", in *SCA 2006 Eurographics / ACM SIGGRAPH Symposium on Computer Animation, Posters & Demos*, Vienna, Austria, 2006, pp. 27–28.
- [11] Thomas Buschmann, Sebastian Lohmeier, Heinz Ulbrich, and Friedrich Pfeiffer, "Dynamics simulation for a biped robot: Modeling and experimental verification", in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, Orlando, Florida, May 2006, pp. 2673–2678.
- [12] David E. Stewart and Jeffrey C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with Coulomb friction.", in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2000, pp. 162–169.
- [13] Eran Guendelman, Robert Bridson, and Ronald Fedkiw, "Nonconvex rigid bodies with stacking", *ACM Trans. on Graphics*, vol. 22, no. 3, pp. 871–878, 2003.
- [14] Harald Schmidl and Victor J. Milenkovic, "A fast impulsive contact suite for rigid body simulation", *IEEE Trans. on Visualization and Computer Graphics*, vol. 10, no. 2, pp. 189–197, 2004.
- [15] Roy Featherstone, *Robot Dynamics Algorithms*, Kluwer Academic, 1987.
- [16] Jeffrey C. Trinkle, Jong Pang, Sandra Sudarsky, and Grace Lo, "On dynamic multi-rigid-body contact problems with Coulomb friction", *Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 77, no. 4, pp. 267–279, 1997.
- [17] John E. Lloyd, "Fast implementation of Lemke's algorithm for rigid body contact simulation", in *Proc. 2005 IEEE Int. Conf. on Robotics and Automation*, Barcelona, Spain, April 2005, pp. 4549–4554.
- [18] Jane Wilhelms, Matthew Moore, and Robert Skinner, "Dynamic animation: interaction and control.", *The Visual Computer*, vol. 4, no. 6, pp. 283–295, 1988.
- [19] Michiel van de Panne, "Sensor-actuator networks", in *Computer Graphics (Proc. of ACM SIGGRAPH 93)*, 1993, pp. 335–342, ACM Press.
- [20] Katsu Yamane and Yoshihiko Nakamura, "Stable penalty-based model of frictional contacts", in *IEEE International Conference on Robotics and Automation*, May 2006, pp. 1904–1909.
- [21] Brian Mirtich, "Rigid body contact: Collision detection to force computation", 1998, Technical Report TR-98-01, MERL.
- [22] A. Chatterjee, "On the realism of complementarity conditions in rigid body collisions", *Nonlinear Dynamics*, vol. 20, pp. 159–168, 1999.
- [23] James Allen Fill and Donniell E. Fishkind, "The Moore-Penrose generalized inverse for sums of matrices", *Jrn. on Matrix Analysis and Applications*, , no. 21, pp. 629–635, 1999.
- [24] Fumio Kanehiro, Hirohisa Hirukawa, and Shuuji Kajita, "OpenHRP: Open architecture humanoid robotics platform", *I. J. Robotic Res.*, vol. 23, no. 2, pp. 155–165, 2004.
- [25] Katsu Yamane and Yoshihiko Nakamura, "Dynamics filter-concept and implementation of on-line motion generator for human figures", *IEEE Trans. on Robotics and Automation*, vol. 19, no. 9, pp. 421–432, June 2003.
- [26] Katsu Yamane and Yoshihiko Nakamura, "Dynamics computation of structure-varying kinematic chains for motion synthesis of humanoid.", in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1999, pp. 714–721.