PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A

Phil. Trans. R. Soc. A (2007) **365**, 79–107
doi:10.1098/rsta.2006.1921
Published online 17 November 2006

The experimental humanoid robot H7: a research platform for autonomous behaviour

By Koichi Nishiwaki[1,*], James Kuffner[2,1], Satoshi Kagami[1,3], Masayuki Inaba[3] and Hirochika Inoue[1]

[1]*Digital Human Research Center, National Institute of Advanced Industrial Science and Technology (AIST), 2-41-6, Aomi, Koto-ku, Tokyo 136-0064, Japan*
[2]*The Robotics Institute, Carnegie Melon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA*
[3]*Graduate School of Information Science and Technology, The University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo 113-8856, Japan*

This paper gives an overview of the humanoid robot 'H7', which was developed over several years as an experimental platform for walking, autonomous behaviour and human interaction research at the University of Tokyo. H7 was designed to be a human-sized robot capable of operating autonomously in indoor environments designed for humans. The hardware is relatively simple to operate and conduct research on, particularly with respect to the hierarchical design of its control architecture. We describe the overall design goals and methodology, along with a summary of its online walking capabilities, autonomous vision-based behaviours and automatic motion planning. We show experimental results obtained by implementations running within a simulation environment as well as on the actual robot hardware.

Keywords: humanoid robot; autonomous behaviour; biped locomotion; motion planning; vision-based control

1. Introduction

The recent rapid progress in the development of commercial prototype humanoid robots around the world has stimulated a renewed interest in biped walking control, robot–human interaction and artificial intelligence research. However, owing to the limited availability, high cost and proprietary nature of most humanoids, the ability to conduct advanced research in these areas is difficult. This is particularly true for academics in a university setting.

This paper summarizes a multi-year project to develop the humanoid robot 'H7' as a research platform for experimental robotics. The overall goal was to provide a research test bed for investigating perception–action coupling and intelligent behaviours. Some of the key design components of H7 include: (i) a full-size body with sufficient degrees of freedom (d.f.) and joint torque for whole-body

* Author for correspondence (k.nishiwaki@aist.go.jp).

One contribution of 15 to a Theme Issue 'Walking machines'.

motions, (ii) standard PC/AT compatible high-performance on-board processing, (iii) RT-Linux operating system for realizing simultaneous low-level and high-level control, (iv) a self-contained system design with on-board power and wireless communication capabilities, and (v) dynamic walking trajectory generation, motion planning and three-dimensional vision functions for implementing complex behaviours.

The rest of the article is organized as follows: §2 gives an overview of the hardware design and software architecture of H7; §3 provides details regarding the online walking control system; §4 contains an overview of the various autonomous motion planning functions; §5 shows some autonomous behaviour experiments realized on H7; and §6 concludes with a summary discussion.

2. Hardware design and software architecture

In selecting the design for H7, we considered a number of desirable characteristics for a research platform for autonomous whole-body behaviour. The primary design characteristics we focused on include the following.

— A human-sized mechanism with sufficient d.f., joint power (torque and speed) and joint motion range for a variety of whole-body motions.
— The ability of parts other than the sole to contact the environment without damaging itself or the environment.
— Visual sensors and distributed tactile sensors for perceiving the surrounding environment.
— High-performance computing resources for environment recognition, dynamic trajectory generation and motion planning.

The motivation for these design choices are based on both practical need and our experience developing earlier prototype humanoids. In particular, H7 is a revised and improved version of the H6 humanoid robot (Nishiwaki *et al.* 2000), which has a similar joint structure and d.f. arrangement. The motor power specifications for H7 were designed to be sufficient for walking as well as standing up from a prone position. Based on the knowledge obtained from the experiments with H6, the motor drivers were newly designed and the selection of motor and gear ratios for H7 was tuned. The outward appearance consists of links with a smooth surface shape that almost entirely encases the actuators, electronics, power supply and wiring. The smooth external geometry is also advantageous for covering the entire body surface with dense tactile sensing elements, such as the various 'artificial skin' prototypes currently under development. PC/AT compatible computer hardware was adopted to provide a standard high-performance computing platform and mounted directly inside the torso. A built-in wireless LAN system and onboard batteries enable the robot to be operated completely without any external cables, which is important for conducting experiments in both general and complex environments. Head-mounted stereo cameras, six-axis force sensors and a gyroscope sensor are the main components of the perception subsystem.

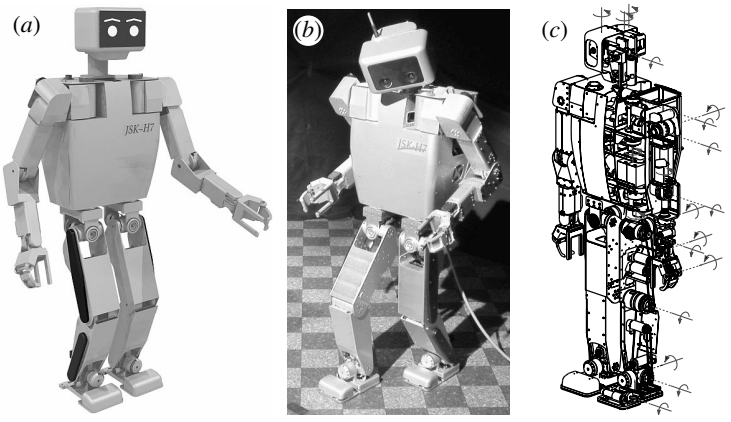

Figure 1. Humanoid robot H7. (*a*) Simulation snapshot, (*b*) photograph and (*c*) mechanism design.

(*a*) *Specifications*

H7 is 1468 mm tall and is 57 kg. It has a total of 30 d.f.: seven for each leg including a 1 d.f. toe joint, seven for each arm including a 1 d.f. gripper and two at the neck (figure 1).

The computing hardware consists of a CPU board with PICMG connector (dual Pentium III 1.4 GHz) mounted inside the torso, and an IEEE 1394 communication board for image capture, a sound board, and I/O boards with D/A converters, A/D converters and pulse counters, all connected to the CPU board via the PCI and ISA bus.

Onboard power is supplied by four lead-acid batteries (12 V, 2.0 Ah) weighing 0.86 kg each, which are mounted inside the torso. Optional Ni-MH battery modules can also be attached to the back (6 kg, 48 V, 6 Ah is the maximum).

A tiny IEEE 802.11b wireless station (58 mm (w) $\times$82 mm (d) $\times$22 mm (h), 110 g) is mounted inside the head. The head is also outfitted with a high-resolution stereo camera module with an IEEE 1394 interface (Videre Mega-D: 1280$\times$1024 pixels, with a field of view of approximately 90°).

Thin and light 6-axis force sensors that we developed specifically for measuring ground reaction forces (Nishiwaki *et al.* 2002) are installed in the feet. Additionally, a gyroscope sensor that outputs three-dimensional attitude and three-dimensional translational acceleration is mounted inside the torso.

(*b*) *Software architecture*

We adopted a centralized system in order to construct a platform where we can easily develop and debug perception–action coupling control schemes. All of the sensor data, such as actuator encoder values, reaction forces and camera images, are directly available to the PC. Control commands are sent from the PC to the motor drivers directly. This requires an operating system in which many different cycle control loops can be executed concurrently (from 1 ms servo loop to higher level trajectory generation and motion planning loop, which have cycle times of several seconds). We adopted RT-Linux (Barabanov 1997; Linux kernel 2.2.18+ RT-Linux 3.0), which is a real-time extension of Linux that can execute 1 ms cycle loops accurately. Since it is based on a Linux core, it is highly suitable

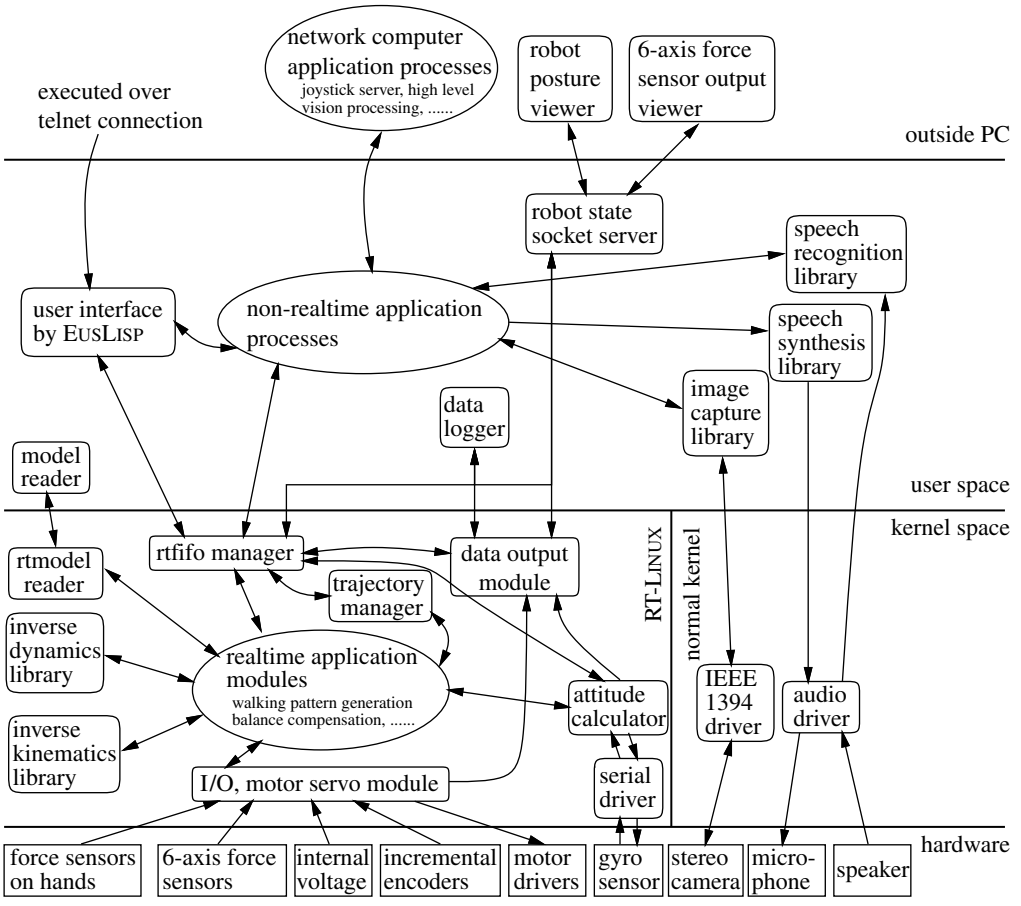

Figure 2. Overview of the software architecture for H7.

as a research platform owing to the wide availability of high-quality, open-source development tools and libraries. All basic operating system functions (file system, networking, etc.) become standard, and it is relatively easy to add new hardware support by writing a custom device driver.

Figure 2 illustrates the overall design of the software architecture. The system consists of real-time modules and user-space programs. Control loops that require accurate execution cycles are implemented as real-time kernel modules, such as the motor servo loop and the online walking control system. Programs and processes that operate over longer control cycles, such as vision processing and motion planning loops, are implemented as user-space program. User-space programs are easier to develop compared with kernel modules and are readily interfaced to the EusLisp (Matsui & Inaba 1990) system which several of our modules used.

3. Online walking control system

This section provides an overview of a humanoid walking control system that generates body trajectories to follow a given desired motion online. A layered software and control architecture is used to aggregate system components and

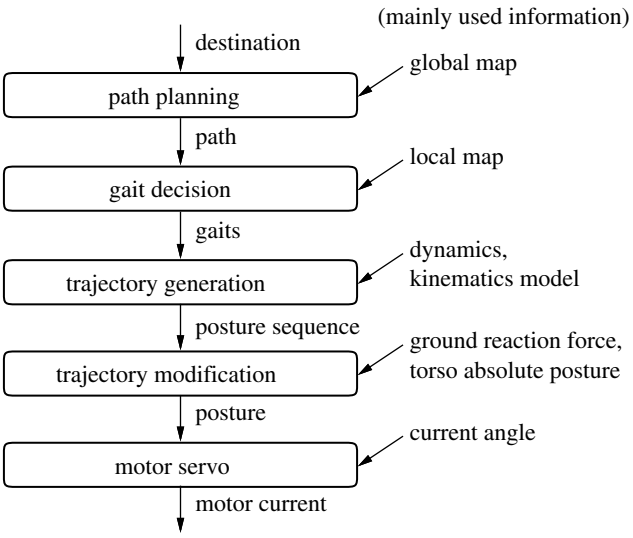

Figure 3. Example of a layered structure for online walking control.

provides a framework for high-level autonomous locomotion behaviours. Walking characteristics such as desired torso movements, upper body posture and step cycles, can be specified and used to generate stable whole-body walking trajectories online. The basic architecture consists of four layers: footstep decision; trajectory generation; trajectory modification from sensor feedback; and joint servo control.

(*a*) *Layered control approach*

The ultimate goal of autonomous locomotion is to enable a robot to navigate to a destination point automatically by acquiring information about the environment and planning a suitable motion to reach the goal. In order to achieve this autonomy, many techniques are required, such as environment map generation, localization, path planning, gait planning, reactive avoidance of obstacles, dynamic stabilization control and motor servo control. These processes must operate online concurrently, despite the fact that they have different control cycles that depend on the calculation time and update cycles of the incoming sensory information.

In order to accommodate these design constraints, we developed a hierarchical architecture that consists of layers of different control cycles. An example of a layered architecture for autonomous walking is shown in figure 3. In this architecture, the processed result of one layer communicates the control value to the next (lower) layer, with higher layers usually having slower control frequencies. Four layers starting from the top are described in this section, including motor servo, trajectory modification, trajectory generation and gait planning. Our implementation enables online walking control that satisfies a given robot translation and rotation with arbitrary upper body posture and step cycles.

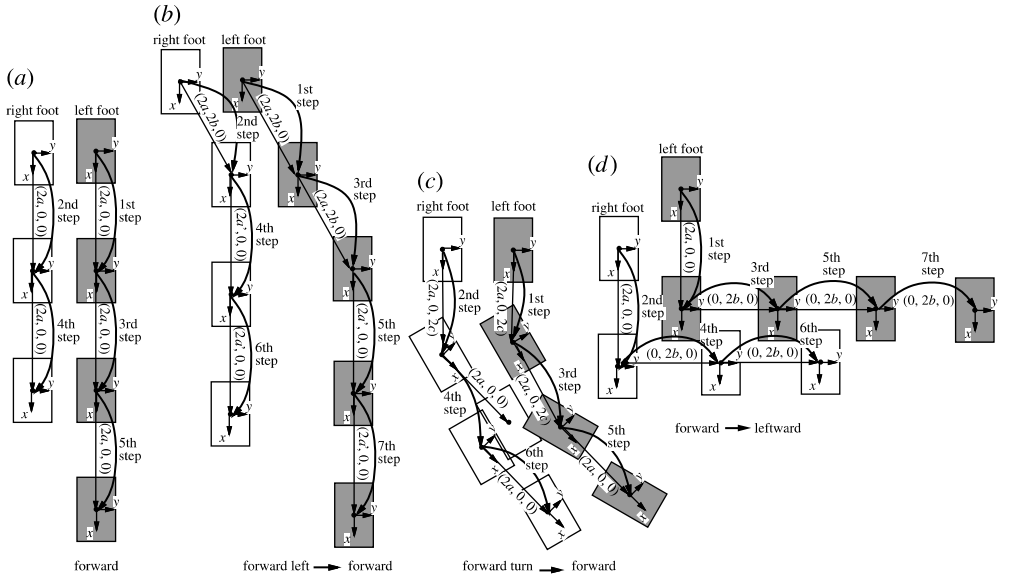

Figure 4. Footstep location selection by transforming desired torso movements into movements of the swing leg foot.

(*b*) *Gait and footstep location selection*

During walking, the movement of the torso is roughly half of that of the swing leg foot on average. Thus, a simple way to decide target footstep locations on level ground is to make the movement of the foot twice the desired torso motion in one step. However, in our experiments, this method turned out to generate unnatural and inefficient footstep locations in many cases (see figure 4 for examples).

We propose a method that calculates footstep landing positions relative to the foot of the supporting leg, given the desired torso movement. Figure 5 shows landing positions calculated in a coordinate system whose origin is fixed at the foot of the supporting leg.

Let the desired torso motion in one step be (x, y, θ), where x is the forward offset; y is the lateral offset; and θ is the counterclockwise orientation. The landing position is calculated as follows: $(x, 2y+w, \theta)$ for the left foot swing leg and $(x, 2y-w, \theta)$ for the right foot. Here, the x-axis is aligned with the forward direction of the supporting leg foot and w is the normal distance of the feet in the lateral direction. Figure 5 shows some examples of generated footprints using this method. Figure 5*a* shows an example where the torso of the robot moves forward a distance a in one step. Figure 5*b* shows an example of the torso rotating by the angle b, but maintaining a fixed position on average. Here, k is the coefficient that increases the minimum distance between the two feet in proportion to the rotation angle to avoid collisions between the legs. In figure 5*c*, the geometrical constraint that prevents the feet from crossing each other sideways results in a maximal torso motion of $c/2$ in the lateral direction. Therefore, the calculated landing position is doubled only for the lateral component. Figure 5*d–g* shows

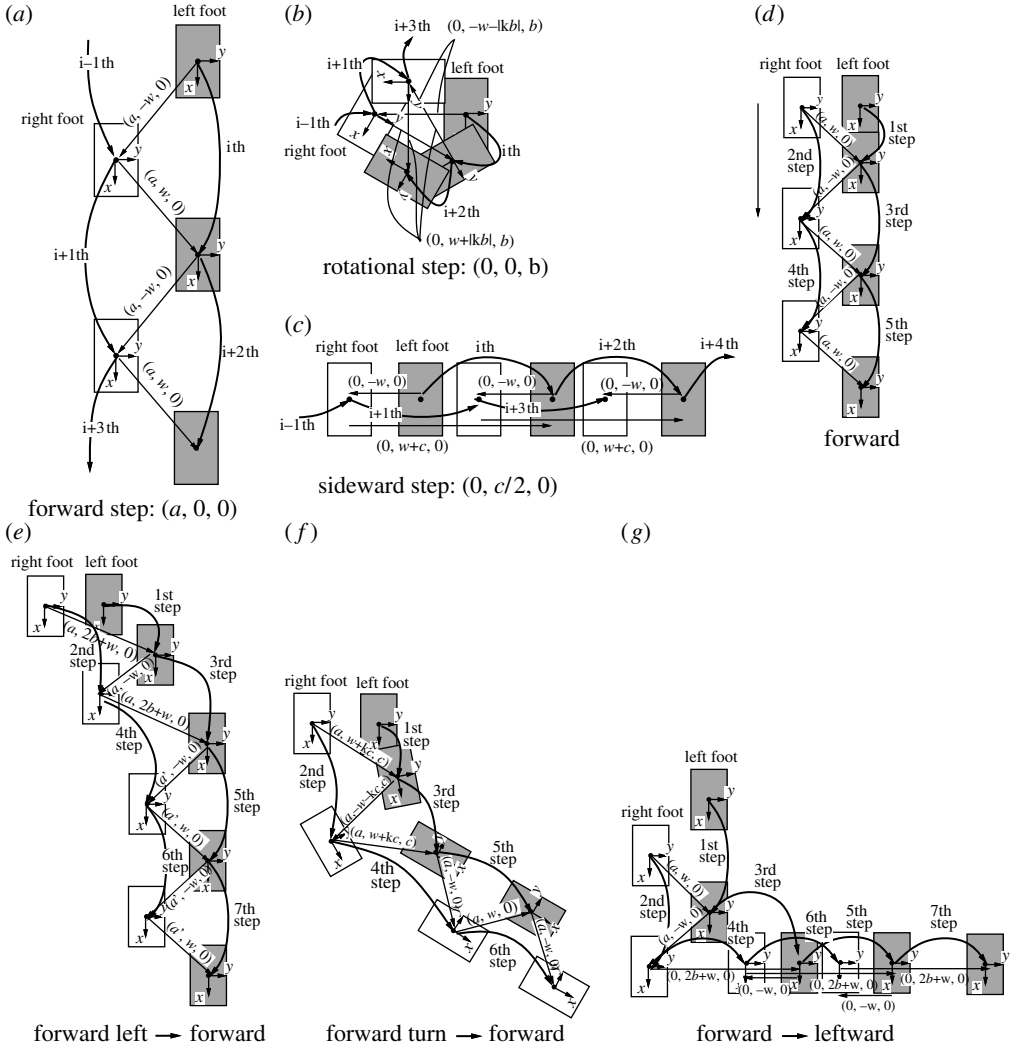

Figure 5. Footstep location selection by transforming desired torso movements into landing positions of the swing leg foot relative to the support leg.

generated footprints for the same torso motions as those given in figure 4. The generated footprints satisfy the desired torso motion on average and satisfy exactly when the same motion command is repeated for more than two steps.

(*c*) *Generating dynamically stable walking trajectories*

This section describes the method we developed for efficiently generating dynamically stable walking trajectories. Specifically, we synthesize online walking patterns based on the zero moment point (ZMP) as the stability criteria. In general, the ZMP trajectory can be analytically derived from the robot motion trajectory. However, synthesizing a robot walking trajectory that satisfies a given ZMP trajectory analytically is difficult, due to the necessity of

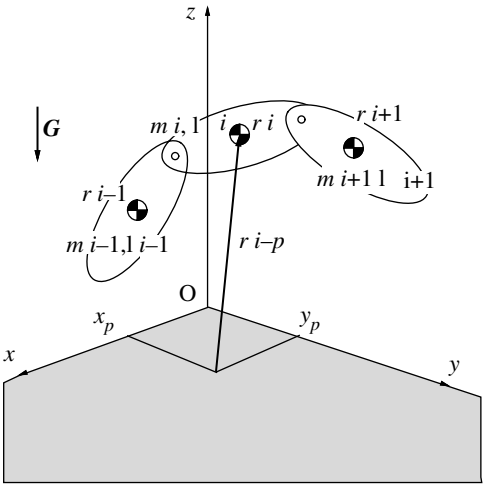

Figure 6. Coordinate system for calculating the ZMP.

solving complex nonlinear second-order differential equations with joint constraints. We have developed an efficient walking trajectory generation method that follows a given input ZMP trajectory. The key to our method is the modification of the torso horizontal trajectory from a given initial trajectory.

Let the ith robot link position, mass, inertia tensor and angular velocity vector be $\boldsymbol{r}_i=(x_i, y_i, z_i)^{\mathrm{T}}$, $\boldsymbol{m}_i$, $\boldsymbol{I}_i$ and $\boldsymbol{\omega}_i$, respectively. Let the x–y plane define the walking surface and let the gravity vector $\boldsymbol{G}$ be the negative z-axis direction ($\boldsymbol{G}=(0,0,-g)^{\mathrm{T}}$) (figure 6). Let $\boldsymbol{P}=(x_p, y_p, 0)^{\mathrm{T}}$ denote the ZMP.

The x component of the ZMP can be calculated from the robot motion as follows (the same for y_p):

$$x_p = \frac{\sum \boldsymbol{m}_i z_i \ddot{x}_i - \sum\{\boldsymbol{m}_i(\ddot{z}_i + g)x_i + (0,1,0)^{\mathrm{T}}\boldsymbol{I}_i\dot{\boldsymbol{\omega}}_i\}}{-\sum \boldsymbol{m}_i(\ddot{z}_i + g)}. \tag{3.1}$$

Let the robot trajectory be denoted as follows:

$$\begin{aligned}\boldsymbol{A}(t) = (&x_1(t), y_1(t), z_1(t), \theta_1(t), \phi_1(t), \psi_1(t),\\ &\ldots, x_n(t), y_n(t), z_n(t), \theta_n(t), \phi_n(t), \psi_n(t)).\end{aligned} \tag{3.2}$$

The ZMP trajectory $\boldsymbol{P}_A(t)=(x_{p_a}(t), y_{p_a}(t), 0)^{\mathrm{T}}$ can be solved using equation (3.1). Now, consider the problem of generating a walking trajectory that follows a desired ZMP $\boldsymbol{P}^*_A(t)$ by only modifying $x_i(t)$, $y_i(t)$ in $\boldsymbol{A}(t)$ to $x'_i(t), y'_i(t)$,

$$x^*_p = \frac{\sum m_i z_i \ddot{x}'_i - \sum\{\boldsymbol{m}_i(\ddot{z}_i + g)x'_i + (0,1,0)^{\mathrm{T}}\boldsymbol{I}_i\dot{\boldsymbol{\omega}}_i\}}{-\sum \boldsymbol{m}_i(\ddot{z}_i + g)}. \tag{3.3}$$

This problem requires solving for x'_i that satisfies equation (3.3) (the same for y'_i). From equations (3.1)–(3.3), we obtain the following equivalence:

$$x^e_p = \frac{\sum \boldsymbol{m}_i z_i \ddot{x}^e_i - \sum \boldsymbol{m}_i(\ddot{z}_i + g)x^e_i}{-\sum \boldsymbol{m}_i(\ddot{z}_i + g)}. \tag{3.4}$$

Here, $x^e_p = x^*_p - x_{p_a}$, $x^e_i = x'_i - x_i$.

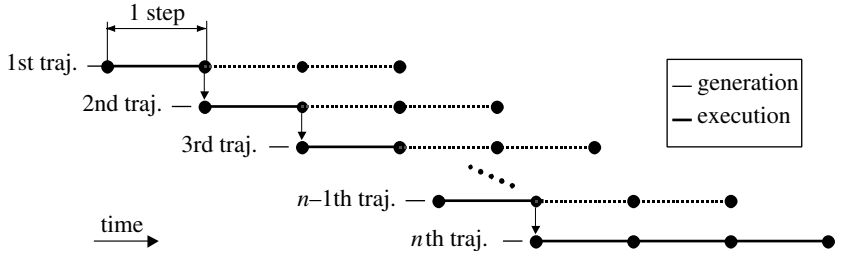

Figure 7. One-step cycle online pattern generation.

Consider $x_i^e = x^e$, i.e. modifying the horizontal position of every link by in same distance. In reality, the feet position cannot be changed relative to the ground. However, the upper body links can be shifted accordingly, up to the limit of the kinematic constraints. Small modifications of the upper body position in the horizontal plane yield proportionally small position changes of the leg links, whose joint values are determined by inverse kinematics. The adjustment lies predominantly in the horizontal plane, with relatively small rotational and vertical components.

By setting $x_i^e = x^e$ to equation (3.4), the following equation is obtained:

$$-\frac{\sum \boldsymbol{m}_i z_i}{\sum \boldsymbol{m}_i(\ddot{z}_i + g)}\ddot{x}^e + x^e = x_p^e. \tag{3.5}$$

In order to solve this numerically, time is discretized on the interval $0, \ldots, t_m$ with time-step Δt. The acceleration at each time $\ddot{x}^e(t_i)$ can be represented as follows:

$$\ddot{x}^e(t_i) = \frac{x^e(t_{i+1}) - 2x^e(t_i) + x^e(t_{i-1})}{\Delta t^2}. \tag{3.6}$$

Then, equation (3.4) can be expressed as trinomial equations. Using the boundary conditions $x^e(0)$, $x^e(t_m)=0$, $x^e(i)$ ($i=1$ to t_m-1) can be obtained. The result is a robot trajectory that satisfies the given ZMP trajectory $P_A^*(T)$. In order to obtain a more accurate trajectory, the calculated trajectory is set as the initial trajectory and this procedure is repeated.

(*d*) *Online walking trajectory generation*

The walking trajectory generation layer is designed to generate stable trajectories that satisfy desired footstep locations, upper body posture and step cycle. Dynamic stability, self-collision and joint performance limitations are considered using a simulation environment in this layer.

Trajectory generation is carried out once for every footstep resulting in a one-step cycle time in this layer. During each cycle, although only a single-step trajectory is absolutely necessary, our system calculates a walking trajectory of three steps into the future. The first two steps are calculated to satisfy the current desired motion, while the third step is used to bring the robot to a halt. In the usual case, only the first step of this three-step trajectory is actually executed by the robot. Instead, a new three-step trajectory is generated and updated during the next cycle (figure 7). Although it may seem that two-thirds of the calculation is wasted, there are key advantages to always executing a trajectory that ends with a dynamically stable stopping motion. Namely, if the

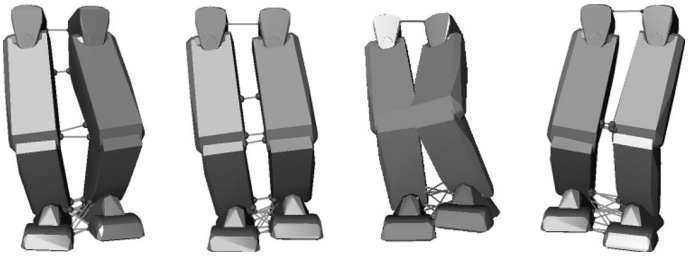

Figure 8. Collision detection between the links of two legs (thighs, forelegs and toes are colliding in the third posture).

calculation or update of the trajectory should fail during the next cycle for some reason, the robot can always be safely brought to a halt simply by continuing to execute the currently available trajectory.

In our system, trajectory generation for the next cycle begins 250 ms before the end of the execution of the first step of the current trajectory. This value is determined by the upper bound on the calculation time with some additional margin. For a three-step stopping trajectory, the longest total motion time is approximately 5.2 s. The balance compensation calculation used to maintain dynamic stability takes *ca* 2.4% of the total motion time using the onboard computer inside the H7 robot (Pentium III 1.1 GHz). Since the dynamics computations consume most of the generation time, it is difficult to repeat dynamically stabilizing calculations during a single cycle. Therefore, additional constraints are considered by the subsequent two steps, including (i) heuristic limitations on the parameters that are used for trajectory generation in order to increase the probability that a realizable dynamically stable trajectory is generated and (ii) validation that the generated trajectory is indeed a realizable one. Other constraints and safety checks are performed after the dynamically stable trajectory generation, including enforcing joint angle and velocity limits, and self-collision detection. For the latter case, we use a fast distance determination method for convex polyhedra in order to conservatively guarantee that the trajectory is free of self-collision (figure 8; Kuffner *et al.* 2002*b*). If a trajectory turns out to result in a self-colliding motion, the update of the trajectory is abandoned and the robot safely comes to a halt by executing the rest of the current trajectory.

(*e*) *Modification of the walking trajectory based on sensor feedback*

The role of the trajectory modification layer is to compensate for disturbances caused by modelling errors, or sudden changes in the environment that cannot be handled by the higher layers. In the case of dynamic stability, if a generated trajectory is executed 'open-loop' without modification, the robot will typically fall down after several steps due to accumulated errors caused by differences between the real world and the modelled world. We have developed three control methods to maintain the dynamic stability of the robot.

— Modification of the horizontal torso position based on the difference between the measured ZMP and the desired ZMP.
— Compensation for deflection around the roll axis of the hip joints based on the output of a gyroscope sensor.

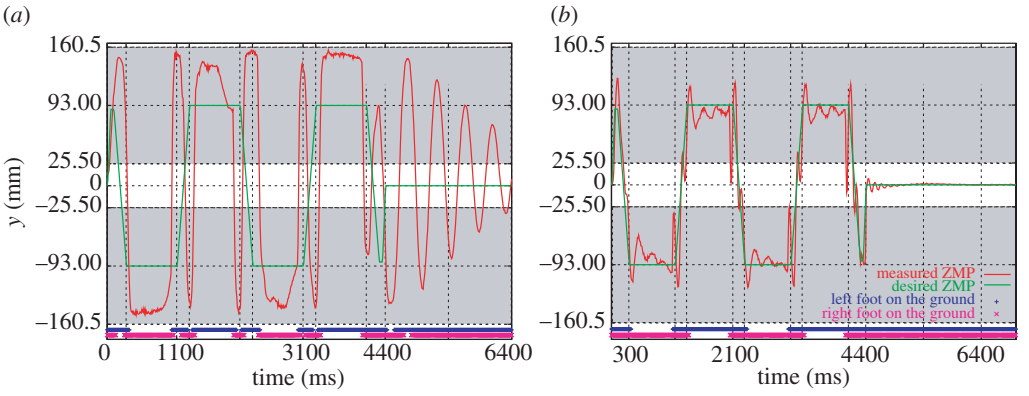

Figure 9. Trajectory of the ZMP in the lateral direction. (*a*) No feedback, (*b*) with sensor feedback.

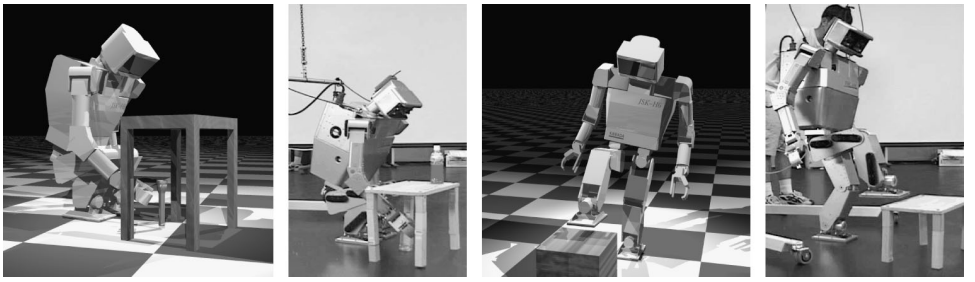

Figure 10. Simulation and video snapshots of planned full-body trajectories.

— Adjusting the joint servo gain according to foot contact timing information in order to reduce the impact of ground reaction forces and internal forces during the dual leg support (DLS) phase.

Figure 9 shows the lateral trajectory of the ZMP and contact state of each foot for four steps of in-place walking. At the start of the motion, the first 300 ms is a DLS phase, with the next 800 ms, a single leg support phase (SLS). During continuous walking, the cycle consists of a 200 ms DLS and 800 ms SLS. When stopping, the robot uses a 300 ms DLS. For this example, the total four-step walking time is 4400 ms. When the three previously mentioned modification techniques are applied, the measured ZMP trajectory follows the desired ZMP trajectory much more closely, and the transitions between contact states are greatly improved. Several autonomous behaviour experiments using our complete online walking control system are presented in §5.

4. Automatic motion planning

This section provides an overview of our efforts to develop practical motion planning methods for humanoid robots for a variety of tasks. Specifically, we have focused on tasks involving navigation, object grasping and manipulation, footstep placement and full-body motions (figure 10). In the latter case, we consider the problem of computing dynamically stable, collision-free trajectories for the entire

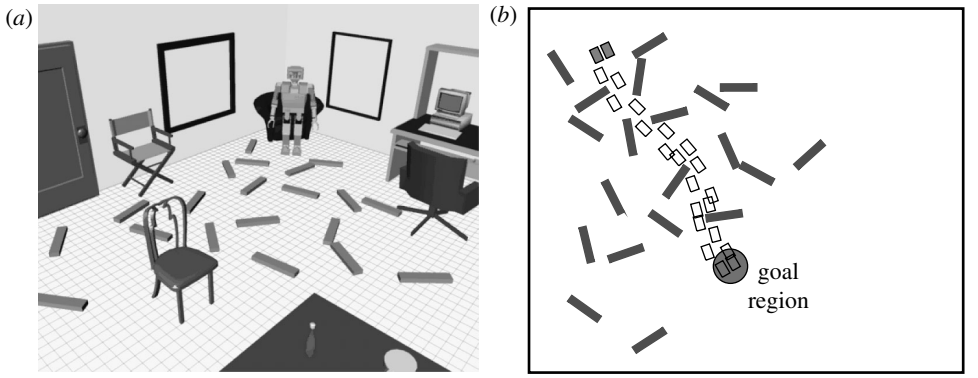

Figure 11. (*a*) Humanoid navigating in a cluttered office. (*b*) Planned footstep locations (top view).

body. In the sections that follow, we describe the algorithms developed for each task and show experimental results obtained by implementations running within a simulation environment as well as on actual humanoid robot hardware.

(*a*) *Footstep planning*

Global path planning and obstacle avoidance strategies for mobile robots and manipulators have a large and extensive history in the robotics literature (see Latombe (1991) and Hwang & Ahuja (1992) for an overview of early work). Global navigation strategies for mobile robots can usually be obtained by searching for a collision-free path in a two-dimensional environment. Owing to the low dimensionality of the search space, very efficient and complete (or resolution complete) algorithms can be employed. For humanoid robots, conservative global navigation strategies can be obtained by choosing an appropriate bounding volume (e.g. a cylinder) and designing locomotion gaits for following navigation trajectories computed by a two-dimensional path planner. However, this always forces the robot to circumvent obstacles. In contrast, legged robots (including biped humanoids) have the unique ability to traverse obstacles by stepping over or upon them. Since reliable walking biped robots have been developed only recently, much less research attention has been focused on this area.

The goal of *footstep planning* is to compute a sequence of footstep placements to navigate to a desired goal location in an obstacle-cluttered environment. Our approach is to build a search tree from a discrete set of feasible footstep locations corresponding to available stepping motions (Kuffner *et al.* 2001; Chestnutt *et al.* 2003). Using standard dynamic programming techniques, optimal sequences of footstep placements can be computed according to encoded heuristics that minimize the number and complexity of the steps taken. Such a strategy can be computed efficiently on standard PC hardware (under 1 s for simple environments and in a few seconds for relatively complex, cluttered environments, as shown in figure 11).

(i) *Biped navigation model*

The biped model comes with a pre-determined set of feasible footstep locations for each foot. For example, figure 12 shows the continuous feasible footstep range $\mathrm{FR}_{\mathrm{right}}$ for the right foot while supported by the left foot, and an example discrete

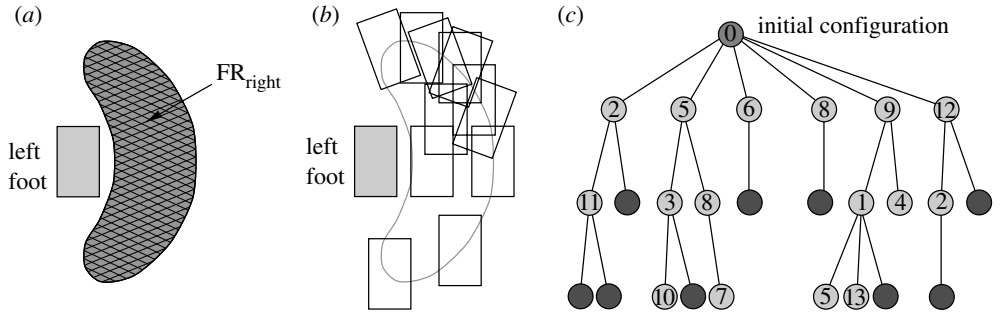

Figure 12. Reachable placement positions for the right foot (*a*) continuous region, and (*b*) discrete placements. (*c*) Search tree with pruned successor states (dark grey) that resulted in bad foot placements or collisions.

set of foot placements. For symmetric bipeds, the placements for the left foot can simply mirror the right-foot placements. In selecting which footstep placements to include in the discrete set used during the search, we chose a distribution of placements along the edge of the reachable region at different relative foot angles as well as a few interior placements to allow the robot to manoeuvre in tight areas. This choice represents a tradeoff between planning performance and generality. The goal is to strike a balance between maximizing the navigation options, while minimizing the total number of discrete placements (the branching factor of the search tree). In our implementation, we selected a total of 15 placements for each foot. In addition to the set of footstep placements, the planner also requires a method to generate dynamically stable motion trajectories for transitioning between them. These trajectories can be either pre-calculated and stored (Kuffner *et al.* 2001) or generated using an online algorithm (Chestnutt *et al.* 2003).

(ii) *Footstep planning algorithm*

The planner accepts as input a discrete set of robot footprint locations, a trajectory generator and a heuristic cost function. Both two- and three-dimensional representations of the robot and environment model can be used for collision checking (see Chestnutt *et al.* 2003). If the planner successfully finds a solution, it outputs a sequence of encoded footstep placements and transitions. A *forward dynamic programming* approach to planning navigation strategies is adopted, which can also be generalized to classic A* (A-star) search. Since an exhaustive search is too expensive, we employ a heuristic evaluation function in order to prune the search tree. Starting from an initial biped configuration Q_{init}, a search tree of possible footstep placements is constructed. The planner maintains a *priority queue* of search nodes containing footstep placements and *cost values*. The cost function $\mathcal{L}(Q)$ defines a simple greedy heuristic

$$\mathcal{L}(Q) = w_{\text{d}} D(N_Q) + w_{\rho} \rho(N_Q) + w_{\text{g}} \mathcal{X}(Q, Q_{\text{g}}).$$

The first two terms define the cost of the path to configuration Q from Q_{init}; $D(N_Q)$ is the depth of the node N_Q in the tree; and $\rho(N_Q)$ is a function that encodes the path 'goodness', such as favouring 'safe' overall foot placements, as well as paths which incur few orientation changes (for detailed example path

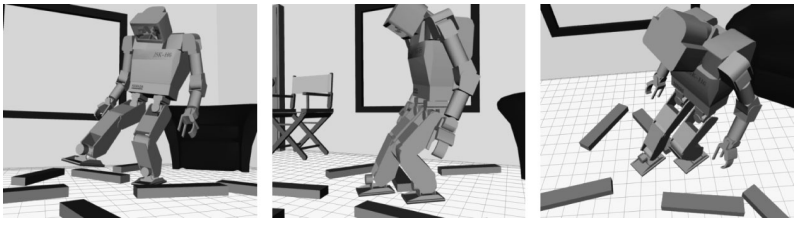

Figure 13. Simulation snapshots during execution of footstep plan.

metrics, see Chestnutt *et al.* (2003)). These terms have the combined effect of favouring paths with fewer steps, as well as slightly favouring paths with long sequences of straight-line steps. The final term represents an estimated cost from the current configuration to the goal region. $\chi(Q, Q_g)$ approximates the minimum number of steps needed to traverse the straight-line distance between the footprint location at Q and a footprint in the centre of the goal region Q_g. Each of the terms is weighted relative to each other by the factors w_d, w_ρ and w_g.

Figure 11 shows a cluttered office in which a model of the humanoid robot must navigate and a top view of a footstep sequence computed to reach a circular goal region in the centre of the room. There were a total of 15 discrete foot placements considered for each foot and a total of 20 floor obstacles. The search tree contained approximately 830 000 nodes. Considering that the number of nodes required for a brute-force, breadth-first search on a footstep sequence length of 18 steps is more than 10^{21}, this is quite satisfactory. The path was computed in approximately 4 s on a 1.6 GHz Pentium4 running Linux. We used a two-dimensional polygon–polygon intersection test for the first phase of collision checking, and the (*V-clip*) library (see Mirtich 1998) for fast minimum distance determination between the obstacles and the convex hull of each leg link for the second phase (figure 13).

(*b*) *Object manipulation*

Manipulation tasks are specified by identifying a target object to be moved and its new location. The motion planning software will then attempt to compute three trajectories: *reach*, position the robot to grasp the object; *transfer*, after grasping, move the object to the target location; and *return*, once the object has been placed at the target location, release it and return the robot to its rest position. In this case, the start and the goal are body postures that must be connected by a path in the configuration space. If a path at each phase is successfully returned by the planner, the robot executes the motion, possibly guided by visual or force feedback. There are many potential uses for such software, with the primary one being a high-level control interface for automatically solving complex object manipulation tasks.

Owing to the complexity of motion planning in its general form (Reif 1979), the use of complete algorithms is limited to low-dimensional configuration spaces. Even single-arm manipulation planning (typically 6–7 d.f.s) presents a computational challenge due to the dimensionality of the search space. Since it is typically impractical to explicitly represent the configuration space, sampling techniques are often used in order to discover free configurations and build a data structure that approximates their connectivity. The problem then becomes how to design practical and efficient sampling schemes. This has motivated the

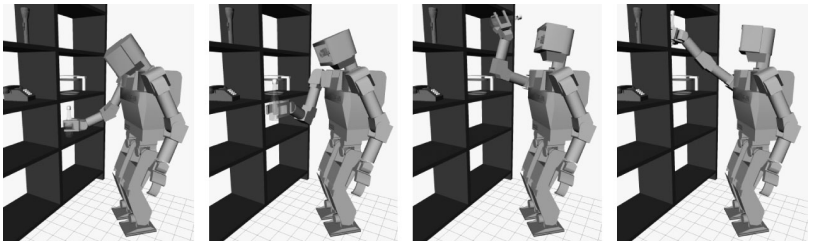

Figure 14. Manipulation planning simulation environment.

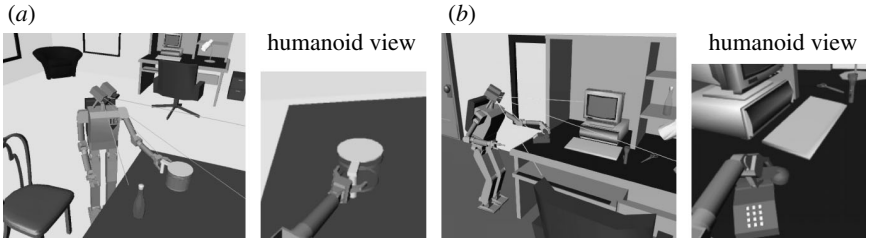

Figure 15. (*a*) Grasping a coffee pot. (*b*) Answering the telephone.

development of numerous planning methods, many of which employ techniques such as randomization (e.g. Barraquand & Latombe 1990; Kavraki *et al.* 1996; Hsu *et al.* 1997; Mazer *et al.* 1998; Kuffner & LaValle 2000), lazy evaluation of collision checking (e.g. Bohlin & Kavraki 2000; Sanchez & Latombe 2002), deterministic sampling (Branicky *et al.* 2001) or a combination of techniques. Although these methods are often heuristic and incomplete, many have been shown to find paths in high-dimensional configuration spaces with high probability.

We have adopted an efficient general path planning algorithm that is well suited for manipulation planning. The algorithm, *RRT-Connect* (Kuffner & LaValle 2000), was originally developed to plan collision-free motions for animated characters in three-dimensional virtual environments (Kuffner 1999). It uses a randomized search strategy based on rapidly exploring random trees (RRTs; LaValle & Kuffner 1999). Distinguishing features of this algorithm include no pre-processing of the workspace (ideal for changing environments), greedy behaviour that solves simple queries very efficiently and uniform coverage of any non-convex space (for details and analysis, see Kuffner & LaValle (2000)).

Combined with an inverse kinematics algorithm, the planner facilitates a task-level control mechanism for planning manipulation motions. Through a graphical user interface, an operator can click and drag an object to a target location and issue a *move* command. Figure 14 shows snapshots of a planned motion for a humanoid repositioning a bottle from the lower shelf to the upper shelf. In the examples shown in figure 15, the simulated vision module is used in order to verify that a particular target object is visible to a virtual humanoid prior to attempting to grasp it. If the object is visible, the manipulation planner is invoked to plan a collision-free path to grasp the object. If the target object is not visible, the humanoid will attempt to reposition itself, or initiate a searching behaviour in an attempt to find the missing object. Additional online experiments using this manipulation planner with stereo vision output is presented in §5.

(c) *Full-body motions*

Automatic, full-body motion planning for humanoid robots presents a formidable computational challenge due to (i) the high number of degrees of freedom, (ii) complex kinematic and dynamic models, and (iii) balance constraints that must be carefully maintained in order to prevent the robot from falling down. We have developed a version of RRT-Connect that automatically generates collision-free, dynamically stable motions from full-body posture goals (Kuffner *et al.* 2000*a*). Obstacle and balance constraints are imposed upon incremental search motions. Provided that the initial and goal configurations correspond to collision-free, statically stable body postures, the path returned by the planner can be smoothed and transformed into a collision-free, dynamically stable trajectory for the entire body.

(i) *Robot model and assumptions*

An approximate model of the humanoid, including the kinematics and dynamic properties of the links, is used along with the following assumptions.

(i) *Environment model*. We assume that the robot has access to a three-dimensional model of the surrounding environment to be used for collision checking.
(ii) *Initial posture*. The robot is currently balanced in a collision-free, statically stable configuration supported by either one or both feet.
(iii) *Goal posture*. A full-body goal configuration that is both collision-free and statically stable is specified. The goal posture may be given explicitly by a human operator or computed via inverse kinematics or other means.
(iv) *Support base*. The location of the supporting foot (or *feet* in the case of dual-leg support) does not change during the planned motion.

(ii) *Full-body trajectory generation*

The key idea of the planning algorithm is to search the space of statically stable configurations ($\mathcal{C}_{\text{stable}}$) for a solution path that also lies within the free configuration space ($\mathcal{C}_{\text{free}}$). Each incremental search motion checks balance constraints while also checking for collisions with obstacles. Rather than picking a purely random configuration as a target for every planning iteration, we pick from a pre-generated set of statically stable postures (i.e. $q_{\text{rand}} \in \mathcal{C}_{\text{stable}}$). For a more detailed explanation, see Kuffner *et al.* (2002*a*).

If successful, the path search phase returns a continuous sequence of collision-free, statically stable body configurations. All that remains is to calculate a final solution trajectory τ that is dynamically stable and collision free. Theoretically, any given statically stable trajectory can be transformed into a dynamically stable trajectory by arbitrarily slowing down the motion. However, we can almost invariably obtain a smoother and shorter trajectory by performing the following two steps.

(i) *Smoothing*. We smooth the raw path by making several passes along its length, attempting to replace portions of the path between selected pairs of configurations by straight-line segments that satisfy both obstacle and

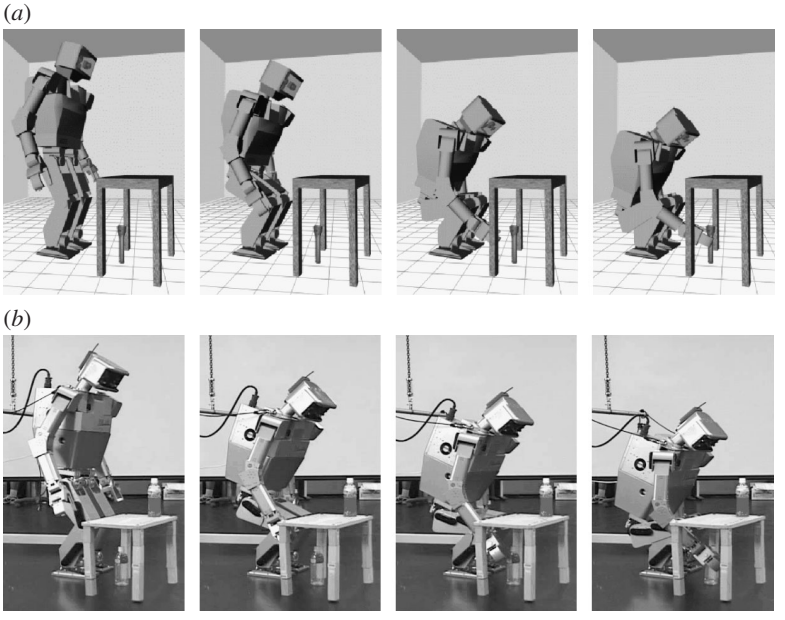

Figure 16. Dynamically stable motion for retrieving an object. (*a*) Simulation. (*b*) Actual hardware.

dynamic balance constraints. This step typically eliminates any potentially unnatural postures along the raw path (e.g. unnecessarily large arm motions). The resulting smoothed path is transformed into an input trajectory using a minimum-jerk model (Flash & Hogan 1985).

(ii) *Filtering.* A dynamics filtering function is used in order to output a final, dynamically stable trajectory. We use the online balance compensation scheme described in Kagami *et al.* (2000*a*), which enforces constraints upon the ZMP trajectory in order to maintain overall dynamic stability. The output configuration of the filter is guaranteed to lie in $\mathcal{C}_{\text{stable}}$. Collision checking is used to verify that the final output trajectory lies in $\mathcal{C}_{\text{free}}$, with the motion made slower in the case of collision.

(iii) *Dynamically stable, collision-free motions*

We have implemented a prototype planner in C++ that runs within a graphical simulation environment. An operator can position individual joints or use inverse kinematics to specify body postures for the virtual robot. The filter function can be run interactively to ensure that the goal configuration is statically stable. After specifying the goal, the planner is invoked to attempt to compute a dynamically stable trajectory connecting the goal configuration to the robot's initial configuration (assumed to be a collision-free, stable posture).

We have tested the output trajectories calculated by the planner online. Figure 16 shows a computed dynamically stable motion for the robot moving from a neutral standing position to a low crouching position in order to retrieve an object from beneath a chair. Figure 17 shows a motion for positioning the right leg above the top of a box while balancing on the left leg. Each of the scenes contains over 9000 triangle primitives. The total wall time elapsed in solving these queries ranges from under 5 s to approximately 1.5 min on a 900 MHz Pentium III running Linux.

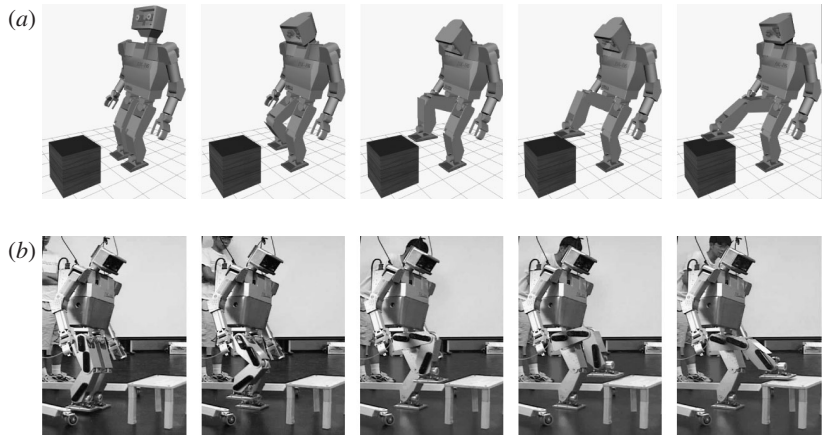

Figure 17. Placing the right foot above the surface of an obstacle while balancing on the left leg. (*a*) Simulation. (*b*) Actual hardware.

5. Autonomous behaviour experiments

(*a*) *Tracking a moving goal with three-dimensional vision*

In order to test our online walking control system, we developed a goal tracking autonomous navigation layer as an example high-level behaviour. The goal tracking behaviour control consists of three parts.

— Stereo vision processing for target detection and three-dimensional position estimation in camera coordinates.
— Planning of the desired future torso movements during one step.
— Camera posture and gaze direction control with self-motion compensation.

(i) *Visual processing*

A stereo camera system mounted on the head is used for tracking a target object of a known colour that represents the navigation goal location. In our experiments, we used a red ball to denote the navigation goal. While the robot and the goal are both moving, colour information from the camera images is used to detect the relative goal direction. If the target is obscured or outside the viewing area, either the most recent detected position may be used or the robot may enter a 'search' mode to locate and reacquire the target.

We developed a real-time depthmap generation algorithm (Kagami *et al.* 2000*b*) to measure the distance to the goal. This method uses four key techniques to achieve high speed and accuracy: (i) recursive (normalized) correlation, (ii) cache optimization, (iii) online consistency checking, and (iv) using the MMX/SSE(R) multimedia instruction set for optimized performance. The final output of this subcomponent is the three-dimensional position of the target relative to the cameras.

Depthmap generation

The correspondence between every pixel from one image to the other is required to generate a depthmap. We assume that the epipolar line is horizontal, thus no

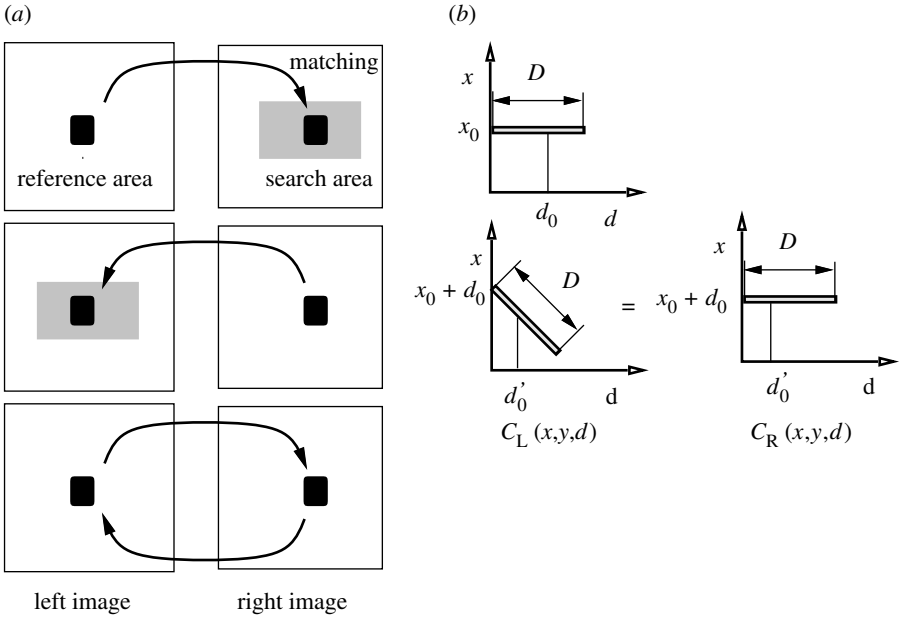

Figure 18. Online consistency checking method.

vertical disparity occurs for two corresponding image regions. Our system uses the recursive correlation method (Faugeras *et al.* 1993) with online consistency checking (Fua 1991; Bolles & Woodfill 1993). We also employ the following three key optimizations: (i) second-level CPU cache utilization, (ii) multimedia instruction set utilization, and (iii) online consistency checking inside recursive correlation technique. We monitored and tuned the CPU cache performance of our recursive correlation implementation. Performance was also optimized using the MMX/SSE/SSE2 multimedia instruction set for the Intel Pentium processor (Kagami *et al.* 2000*b*). These instructions are single instruction multidata (SIMD) and result in 64/128bit parallel calculations. In particular, normalized correlation with reciprocal and reciprocal square root instructions can be calculated approximately five times faster than optimized assembly code generated from standard C source.

Online consistency checking

Stereo matching suffers fundamentally from occlusion problems. The correlation calculation computes the best matching region obtained from candidates. However, a suitable matching region may not be found in the case of an occlusion or on image regions with a lack of texture or other distinguishing features. Thus, noisy or otherwise unreliable matches can result from correlation. Several methods have been proposed to obtain more reliable matching. We have adopted a consistency checking method that proceeds as follows (figure 18*a*):

(i) a reference region (region A) is selected from the left image, and the right image is searched for the best matching region (region B),

(ii) region B is set as the reference region, and the left image is searched in the same way to select the best matching region (region C), and
(iii) if regions A and C are the same, the matching between A and B is considered reliable.

This consistency checking method can be implemented inside the inner loops of the recursive correlation calculation, so that no additional memory is required. Once the correlation value is calculated locally, the first two steps of the consistency check can be calculated simultaneously, followed by the third step that calculates the best match. Figure 18*b* illustrates the process.

Online experiments

Experimental results have demonstrated that our proposed method can calculate a 280×200 depthmap at a rate of 30 Hz from 320×240 input images using the onbody processors of H7. The multimedia instruction set was implemented on gas-2.9.5, with the CMOS high-resolution stereo cameras connected to the CPU via IEEE1394. The output accuracy depends on (i) the lens angle, (ii) the baseline length, (iii) the photosensor size and resolution, and (iv) the distance to the target. The lens intrinsic parameters were calibrated using the improved Tsai method (Tsai 1986) using 500 known points in the scene.

We examined the output accuracy of the stereo depth calculation from 50 to 250 cm and compared them to ground truth. We determined that the onbody system implementation accuracy was approximately 1 cm in (i) 80°, (ii) 9 cm, (iii) 14 mm (2/3 inch diagonal), 320 pixel, and (iv) 1 m. This level of accuracy and calculation time appears to be reasonably sufficient for a human-size humanoid robot to sense an unknown object shape for online grasping and manipulation, and for coarse estimation of the terrain geometry for online navigation and footstep planning.

(ii) *Planning of the torso motion vector*

When walking, the torso of a biped robot does not only move at the specified speed and direction. Rather, it may move in any direction at varying speeds depending upon the compensation motions needed in order to maintain dynamic stability. Thus, using a local coordinate system fixed to the robot (such as the torso origin) is inconvenient for planning movement. Instead, we plan the desired torso motion in world coordinates, which is also convenient for combining the knowledge of the target motion and any stored map information.

Delays due to vision processing must also be taken into account when planning torso movements. The delay between image observation time and the time the target tracking results are available was not negligible in our system. Experiments have shown this delay to be roughly 270–300 ms, so when calculating the goal position in world coordinates, we use the approximate camera position at image capture time (before the start of vision processing) in order to compensate for the time lag in sensing the target location.

The desired position to which the robot should navigate towards is converted to a forward distance a and lateral distance b. A continuum of candidate positions for the robot destination point to reach a fixed distance from the

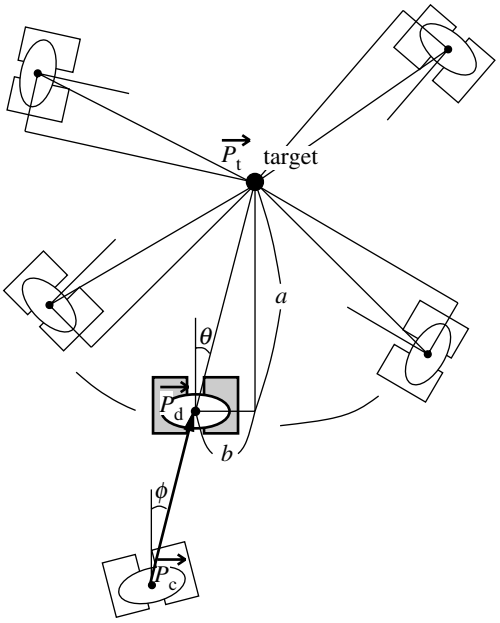

Figure 19. Determining the desired torso motion.

target is shown in figure 19. One destination point is chosen from among the candidates by calculating the point on the line connecting the target and the torso position at the end of the currently executing step ($\bar{\boldsymbol{P}}_{\mathrm{c}}$ in figure 19). The desired torso translation for the next single step is given as $\bar{\boldsymbol{P}}_{\mathrm{d}} - \bar{\boldsymbol{P}}_{\mathrm{c}}$. However, this translational component of the desired motion vector is then projected to lie within the realizable region according to the footprint planning layer. The desired orientation of the torso is expressed as a vertical rotation angle (ψ) given by

$$\psi = \phi - k\theta, \quad k = |\bar{\boldsymbol{P}}_{\mathrm{t}} - \bar{\boldsymbol{P}}_{\mathrm{d}}|/|\bar{\boldsymbol{P}}_{\mathrm{t}} - \bar{\boldsymbol{P}}_{\mathrm{c}}|.$$

Here, k is the coefficient used to prevent the robot from rotating too quickly and moving obliquely when the robot is far from the destination point. The angle ψ is also limited to a realizable range by the footprint planning layer.

(iii) *Camera posture control*

In order to keep the target near the centre of the camera field of view, the head posture of the robot is adjusted online. The head posture of H7 is controlled by pan and tilt joints at the neck. In order to compensate for self-motion, we implemented feed-forward control of the camera gaze direction towards a fixed point in world coordinates. During walking, this is accomplished by calculating the torso position resulting from the trajectory modification layer. The feed-forward control cycle runs at 1 kHz, while the vision process that updates the desired gaze point runs at approximately 10 Hz.

(iv) *Software system*

An overview of the software system and components used in the moving ball target tracking experiment is shown in figure 20. Colour segmentation

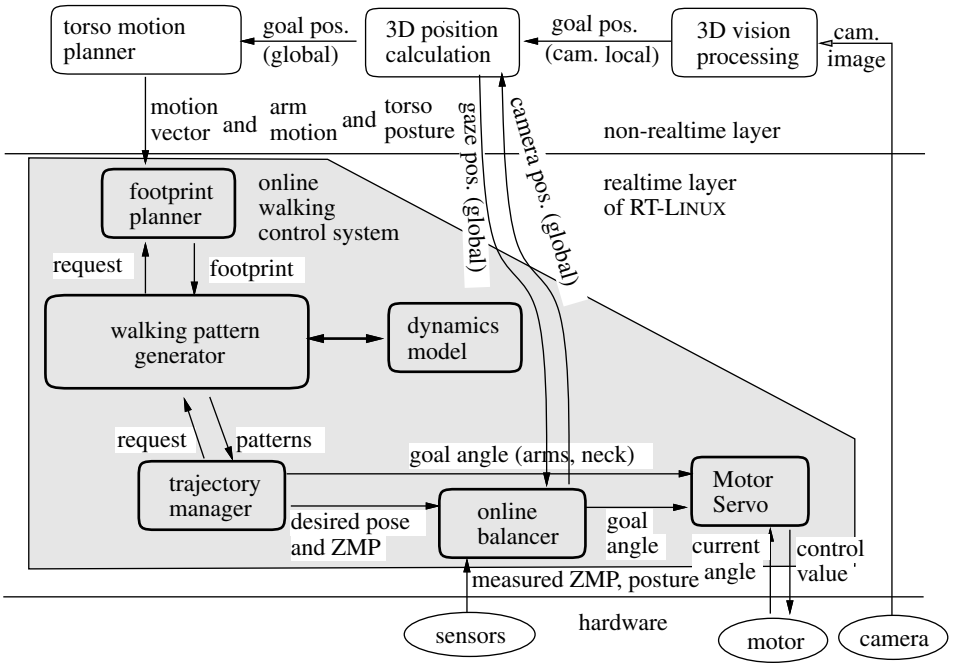

Figure 20. Overview of software components for target tracking while walking.

Figure 21. Snapshots of H7 tracking and following a moving ball target.

and thresholds were used to detect the direction of a moving pink ball. Snapshots taken during an example run of the experiment are shown in figure 21.

Online navigation with footstep planning

In order for bipeds to use their full autonomous navigation capability, dense three-dimensional surface data are needed in order to facilitate footstep planning. In this set of experiments, we connected three-dimensional image sequences to obtain the camera six-dimensional motion and dense three-dimensional environment terrain information. Our method consists of three key

components: (i) stereo depthmap computation, (ii) three-dimensional flow calculation by tracking raw image features, and (iii) 6 d.f. camera motion estimation by a RANdom SAmpling Consenus (RANSAC). We examined and evaluated our method in a motion-capture (MOCAP) environment, so that ground truth data would be available for comparison and evaluation. The system was implemented and tested onboard the H7 robot and achieved an approximately 10 Hz cycle time.

(i) *Conversion matrix calculation*

Assume that the world is rigid and there are no moving objects. If the camera at time t obtains a depthmap $D_t(x, y, z)$ at coordinates W_t, then the coordinate conversion matrix M_{t-1}^t can be derived using the rotation matrix R and translation matrix T as follows:

$$D_{t-1} = R \cdot D_t + T. \tag{5.1}$$

The dimension of the problem is 6 d.f., thus theoretically only three corresponding points in D_t and D_{t-1} are required to calculate R and T. However, there are many errors in both feature tracking and the depthmap calculation. In order to minimize the error, the following equation is used:

$$\min \sum_{j=1}^{n} \| {}^{i-1}C_j - (R \cdot {}^{i}C_j + t) \|^2. \tag{5.2}$$

Here, ${}^{t}C_1, \ldots, {}^{t}C_n$ are the feature points in D_t. We adopted the closed form solution to this problem using a quaternion formulation (Horn 1987).

(ii) *Error checking method*

There are several possible sources of errors, including (i) stereo depth calculation and (ii) feature tracking, after calibrating stereo cameras. Since we assume that world is rigid, any two point sets in time t, $t-1$ satisfy (figures 22 and 23)

$$\| {}^{t}C_i - {}^{t}C_j \| = \| {}^{t-1}C_i - {}^{t-1}C_j \|. \tag{5.3}$$

(iii) *Conversion matrix estimation using RANSAC*

To minimize the influence of errors and noise, we adopted a RANSAC method to estimate M_{t-1}^t. From among N features, we select n features, estimate M_{t-1}^t and calculate the remaining error E using the obtained matrix. This procedure is iterated for a number of times and the best M_{t-1}^t that yields the minimum error is selected.

Figure 22 shows some example experimental results. Figure 22*a* shows the raw three-dimensional flow obtained. Long lines indicate failed feature tracking results. Figure 22*b* shows a least squares-based result for determining a conversion matrix M_{t-1}^t, and the remaining error is indicated by grey lines. Figure 22*c* shows the result after omitting features that do not satisfy rigid body transformation error. Finally, figure 22*d* shows the result using RANSAC, illustrating the relatively small amount of remaining error.

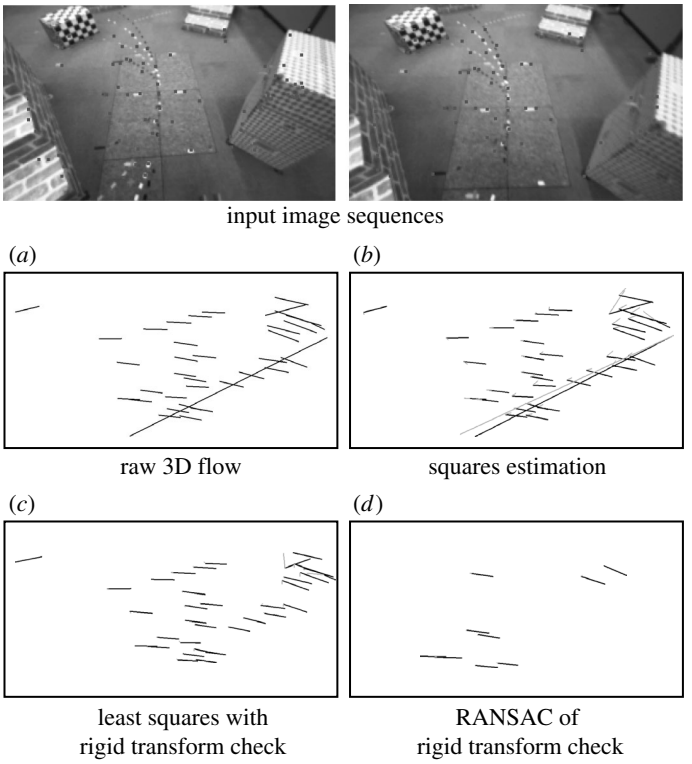

Figure 22. Three-dimensional motion estimation and error.

(iv) *Error minimization in local frames*

Global error minimization is expensive and impractical, given the current computing resources. However, if only two consecutive frames are used, small errors quickly accumulate and the obtained three-dimensional map becomes inaccurate and unusable for robot navigation. Thus, we conducted experiments using a local minimization method.

At the feature tracking stage, we attempt to maintain tracking unless (i) the correlation error value becomes too large or (ii) the rigid body assumption is not satisfied. We obtain the resulting series of feature points $(t-n)C_i$, $(t-n+1)C_i$, …, $(t)C_i$. Then, we can compute the relative transformation matrix between two non-adjacent frames and compare the error terms E to determine which matrix will be used. An example of a three-dimensional scene recovery experiment is shown in figure 24.

(v) *Online mapping and footstep planning*

We examined the integrated system onboard the humanoid robot H7. Figure 25 shows the experimental setup and robot motion and generated three-dimensional map. In the final stage, the complete three-dimensional map is projected down to a 2.5D height map and footsteps from the current robot

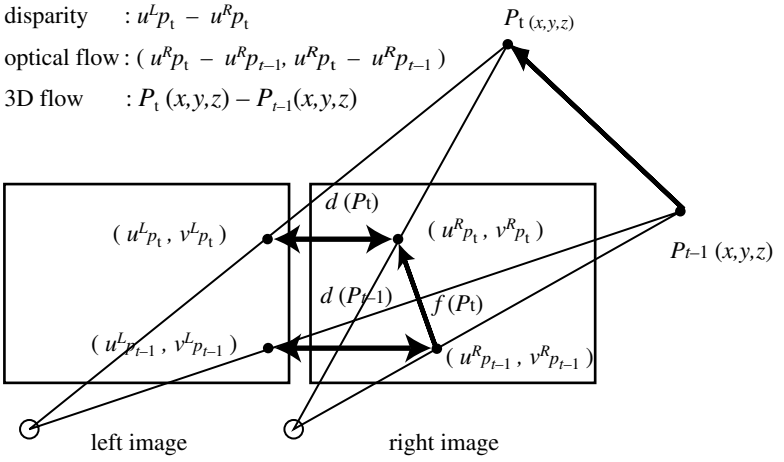

Figure 23. Three-dimensional flow calculation.

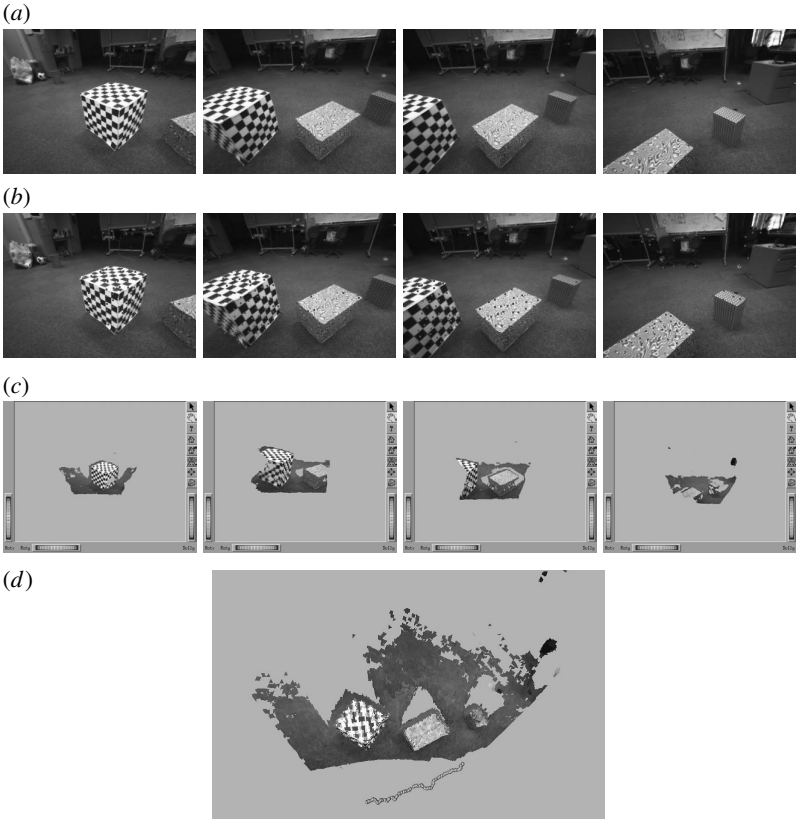

Figure 24. Three-dimensional scene recovery experiment. (*a*) Raw input images. (*b*) KLT tracking feature points. (*c*) One-shot 3D scene from depth map with texture mapping. (*d*) Resulting 3D scene with estimated camera motion.

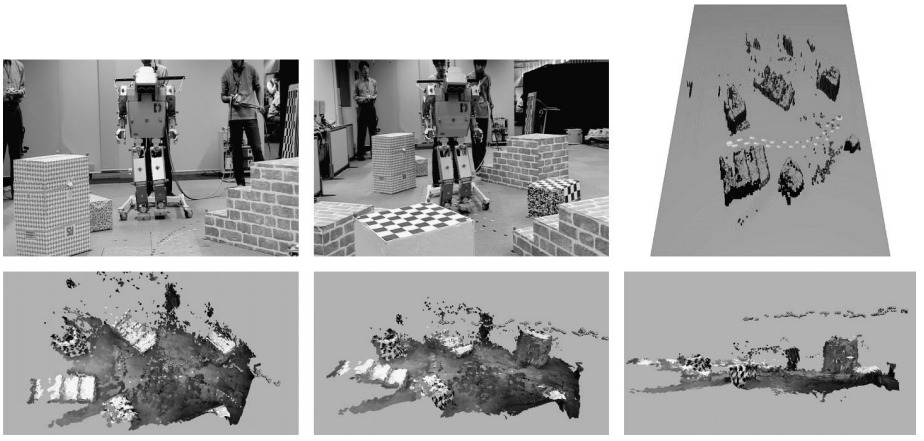

Figure 25. H7 Online mapping and footstep planning.

position to the given goal location are planned. Figure 25 shows the robot avoiding a previously unknown obstacle and reaching to the goal location.

Vision-based object manipulation

In order to use depthmap output for object grasping tasks, accurate and high-speed, dense three-dimensional environment data are required. Let us assume an arm length and hand proportion of a human-size humanoid robot to be approximately 60 and 15 cm, respectively. We considered object grasping tasks that can be accomplished with 1 cm accuracy at 1 m distance at a rate of 10 Hz.

Stereo methods have a reciprocal relationship between distance and resolution. Therefore, results obtained by moving cameras at different distances cannot be directly merged. Using a volumetric representation, the world is discretized into a voxel grid from which surface meshes are extracted. We adopted the Marching Cubes algorithm (Lorensen & Cline 1987) in order to generate volumetric mesh models. The mesh vertices are restricted to lie on grid edges, and each vertex has an associated scalar value that has positive sign when it is outside the adjacent surface and negative sign when it is inside. If the states of adjacent vertices are opposite, the surface of the object intersects the edge between them. We calculate the signed distance of each voxel to the nearest surface along the view line (Curless & Levoy 1996). We accumulate this model incrementally and probabilistically, and obtain an integrated world model (Sagawa *et al.* 2000). In our experiments, a 2 cm voxel size was used.

Let $v_j|(1<j<8)$ denote a voxel vertex and $Z(v_j)$ denote the signed distance of that vertex to the surface. When $Z(v_j)|(1<i<M,1<j<8$ is computed for a M depthmaps, with a weighted average of these signed distances as the result of merging M range images to calculate the final signed distance $V(\boldsymbol{v})$ given by

$$V(\boldsymbol{v}) = \sum_i w_i(\boldsymbol{v}) Z_i(\boldsymbol{v}). \tag{5.4}$$

Here, w is a probabilistic weighting value reciprocal to the distance from the camera to the surface, since the accuracy is inversely proportional to distance.

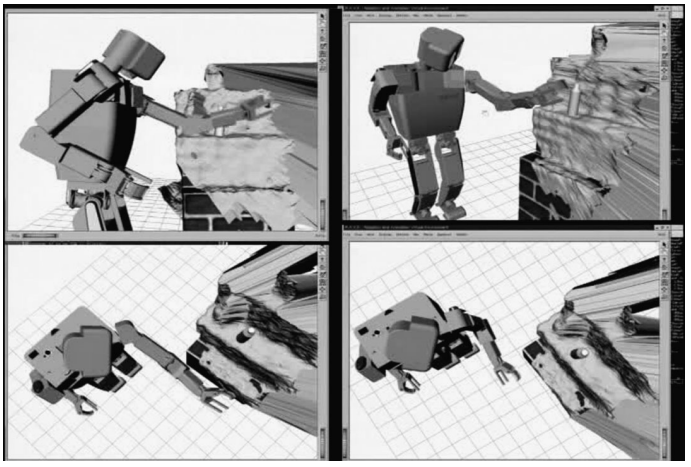

Figure 26. Three-dimensional vision-based arm motion planning.

Incremental updates to the mesh model $V(\boldsymbol{v})$ are given by following equations:

$$V_M(\boldsymbol{v}) = \frac{W_{M-1}(\boldsymbol{v})\,V_{M-1}(\boldsymbol{v}) + w_M(\boldsymbol{v})Z_M(\boldsymbol{v})}{W_{M-1}(\boldsymbol{v}) + w_M(\boldsymbol{v})}, \tag{5.5}$$

$$W_M(\boldsymbol{v}) = W_{M-1}(\boldsymbol{v}) + w_M(\boldsymbol{v}).$$

Here, $W_M(\boldsymbol{v})$ denotes the mesh model size.

(i) *Online humanoid arm motion planning*

Finally, we show results obtained using our complete integrated vision-based environment modelling and RRT-based path planning modules used for online arm trajectory planning for grasping objects of known geometry in unknown environments.

Figure 26 illustrates the robot geometry with respect to the obtained environmental mesh model. In this experiment, the target object (bottle) shape is known *a priori* and an operator must designate the goal object. The vision system yielded a model with approximately 1 cm accuracy. Figure 27 shows the resulting planned bottle-grasping motion for the arm. Although, this example was executed open-loop, H7 was still able to successfully grasp the bottle. The total average calculation time for this problem took approximately 1 s using the onbody processor (minimum 0.3 s, maximum 18.2 s).

6. Summary and discussion

As humanoid robotics technology enters mainstream society during the next several decades, safe operation and autonomy will be of highest importance. The development of general-purpose autonomous humanoid robots represents a very challenging research area, with exciting potential. We have presented an overview of the hardware and software design of our autonomous humanoid research platform H7. We selected standard hardware (PC/AT) and software (RT-Linux OS) components in our design to facilitate easy research and development. We also

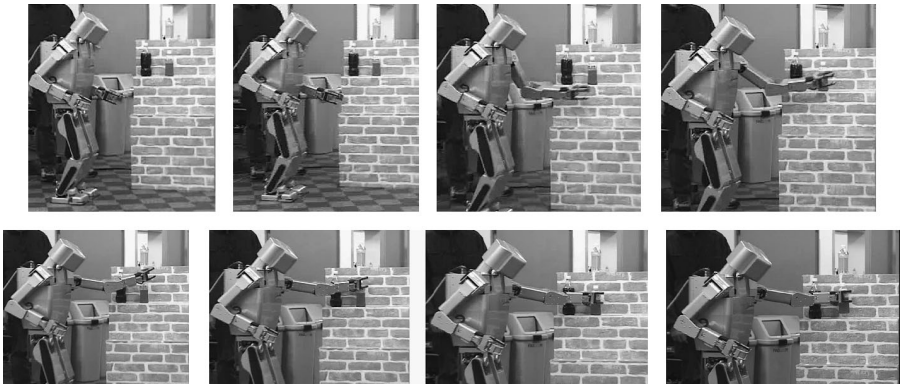

Figure 27. H7 grasping a bottle.

introduce a layered control architecture for developing and integrating complex high-level behaviour controllers with online walking trajectory generation for autonomous locomotion. We have given a brief overview of some our efforts to develop practical motion planning software for humanoids performing a variety of tasks. Using a graphical simulation environment, sophisticated motion generation algorithms can be efficiently developed and debugged, reducing the costs and safety risks involved in testing software for humanoid robots. Furthermore, we hope that through open designs such as those we have developed with H7, the current and the future capabilities of humanoid and other complex robotic systems can be improved.

This research was supported in part by the Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Research for the Future (JSPS-RFTF96P00801), JSPS Grants-in-Aid for Scientific Research (13355011) and a JSPS Postdoctoral Fellowship for Foreign Scholars in Science and Engineering.

References

Barabanov, M. 1997 *A linux-based real-time operating system*, Master's thesis, New Mexico Institute of Mining and Technology, Socorro, NM.

Barraquand, J. & Latombe, J.-C. 1990 Robot motion planning: a distributed representation approach. *Int. J. Robot. Res.* **10**, 628–649.

Bohlin, R. & Kavraki, L. 2000 Path planning using lazy PRM. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*.

Bolles, R. & Woodfill, J. 1993 Spatiotemporal consistency checking of passive renge data. In *Robotics research: the Sixth Int. Symp. International Foundation for Robotics Research* (ed. T. Kanade & R. Paul).

Branicky, M. S., LaValle, S. M., Olson, K. & Yang, L. 2001 Quasi-randomized path planning. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*.

Chestnutt, J., Kuffner, J., Nishiwaki K. & Kagami S. 2003 Planning biped navigation strategies in complex environments. In *Proc. IEEE Int. Conf. on Humanoid Robotics (Humanoids'03)*.

Curless, B. & Levoy, M. 1996 A volumetric method for building complex models from range images. In *Computer Graphics (SIGGRAPH '96 Proceedings)*. **30** (Annual Conference Series), pp. 303–312.

Faugeras, O. *et al.* 1993 Real time correlation-based stereo: algorithm, implementations and applications. *Technical Report N° 2013*, INRIA.

Flash, T. & Hogan, N. 1985 The coordination of arm movements: an experimentally confirmed mathematical model. *J. Neurosci.* **5**, 1688–1703.

Fua, P. 1991 A parallel stereo algorithm that produces dense depth maps and preserves images features. *Machine Vis. Appl.* **6**, 35–49. (doi:10.1007/BF01212430)

Horn, B. K. P. 1987 Closed-form solution of absolute orientation using unit quaternions. *Opt. Soc. Am. A* **4**, 629.

Hsu, D., Latombe, J.-C. & Motwani, R. 1997 Path planning in expansive configuration spaces. *Int. J. Comput. Geomet, Appl.* **9**, 495–512. (doi:10.1142/S0218195999000285)

Hwang, Y. K. & Ahuja, N. 1992 A potential field approach to path planning. *IEEE Trans. Robot Autom.* **8**, 23–32. (doi:10.1109/70.127236)

Kagami, S., Kanehiro, F., Tamiya, Y., Inaba, M. & Inoue, H. 2000*a*. AutoBalancer: an online dynamic balance compensation scheme for humanoid robots. In *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*.

Kagami, S., Okada, K., Inaba, M. & Inoue, H. 2000*b*. Design and implementation of onbody real-time depthmap generation system. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, pp. 1441–1446.

Kavraki, L., Švestka, P., Latombe, J. C. & Overmars, M. H. 1996 Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Trans. Robot. Autom.* **12**, 566–580. (doi:10.1109/70.508439)

Kuffner, J. 1999 *Autonomous agents for real-time animation*, Ph.D. thesis, Stanford University, Stanford, CA.

Kuffner, J. & LaValle, S. 2000 RRT-connect: an efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*.

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

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

Kuffner, J., Nishiwaki, K., Kagami, S., Kuniyoshi, Y., Inaba, M. & Inoue, H. 2002*b*. Self-collision detection and prevention for humanoid robots. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, pp. 2265–2270.

Latombe, J. C. 1991 *Robot motion planning*. Boston, MA: Kluwer Academic Publishers.

LaValle, S. & Kuffner, J. 1999 Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*.

Lorensen, W. E. & Cline, E. 1987 Marching cubes: a high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)* **21** (Annual Conference Series), pp. 163–169.

Matsui, T. & Inaba, M. 1990 EusLisp: an object-based implementation of Lisp. *J. Inf. Process.* **13**, 327–338.

Mazer, E., Ahuactzin, J. M. & Bessière, P. 1998 The Ariadne's clew algorithm. *J. Artif. Intell. Res.* **9**, 295–316.

Mirtich, B. 1998 VClip: fast and robust polyhedral collision detection. *ACM Trans. Graph.* **17**, 177–208. (doi:10.1145/285857.285860)

Nishiwaki, K., Murakami, Y., Kagami, S., Kuniyoshi, Y., Inaba, M. & Inoue, H. 2002 A six-axis force sensor with parallel support mechanism to measure the ground reaction force of humanoid robot. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, pp. 2277–2282.

Nishiwaki, K., Sugihara, T., Kagami, S., Kanehiro, F., Inaba, M. & Inoue, H. 2000 Design and development of research platform for perception-action integration in humanoid robot: H6. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robot. & Sys. (IROS)*, vol. 1, pp. 88–95.

Reif, J. H. 1979 Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symp. on Foundations of Computer Science (FOCS)*, pp. 421–427.

Sagawa, R., Okada, K., Kagami, S., Inaba, M. & Inoue, H. 2000 Incremental mesh modeling and hierarchical object recognition using multiple range images. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robot. & Sys. (IROS)*, vol. 1, pp. 88–95.

Sanchez, G. & Latombe, J. 2002 On delaying collision checking in prm planning—application to multi-robot coordination. *Int. J. Robot. Res.* **21**, 5–26. (doi:10.1177/027836402320556458)

Tsai, R. Y. 1986 An efficient and accurate camera calibration technique for 3D machine vision. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 364–374.