Motion Planning for Humanoid Robots Under Obstacle and Dynamic Balance Constraints

James Kuffner, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, Hirochika Inoue
Dept. of Mechano-Informatics
The University of Tokyo
{kuffner,nishi,kagami,inaba,inoue}@jsk.t.u-tokyo.ac.jp

Abstract

We present an approach to path planning for humanoid robots that computes dynamically-stable, collision-free trajectories from full-body posture goals. Given a geometric model of the environment and a statically-stable desired posture, we search the configuration space of the robot for a collision-free path that simultaneously satisfies dynamic balance constraints. We adapt existing randomized path planning techniques by imposing balance constraints on incremental search motions in order to maintain the overall dynamic stability of the final path. A dynamics filtering function that constrains the ZMP (zero moment point) trajectory is used as a post-processing step to transform statically-stable, collision-free paths into dynamically-stable, collision-free trajectories for the entire body. Although we have focused our experiments on biped robots with a humanoid shape, the method generally applies to any robot subject to balance constraints (legged or not). The algorithm is presented along with computed examples using the humanoid robot "H6".

1 Introduction

Recently, significant progress has been made in the design and control of humanoid robots, particularly in the realization of dynamic walking in several full-body humanoids [11, 37, 27]. As the technology and algorithms for real-time 3D vision and tactile sensing improve, humanoid robots will be able to perform tasks that involve complex interactions with the environment (e.g. grasping and manipulating objects). The enabling software for such tasks includes motion planning for obstacle avoidance, and integrating planning with visual and tactile sensing data. To facilitate the deployment of such software, we are currently developing a graphical simulation environment for testing and debugging [19].

This paper presents an algorithm for automatically generating collision-free dynamically-stable motions from full-body posture goals. It expands upon the preliminary algorithm developed in [21], and has been tested and verified on the humanoid robot hardware platform "H6". Our approach is to adapt techniques from an existing, successful path planner [20] by imposing balance constraints upon incremental motions used during the search. Provided the initial and goal configurations correspond to collision-free, statically-stable body postures, the path returned by the planner can be transformed into a collision-free and dynamically-stable trajectory for the entire body. To the best of our knowledge, this paper represents the first general motion planning algorithm for humanoid robots that has also been experimentally confirmed on real humanoid robot hardware.

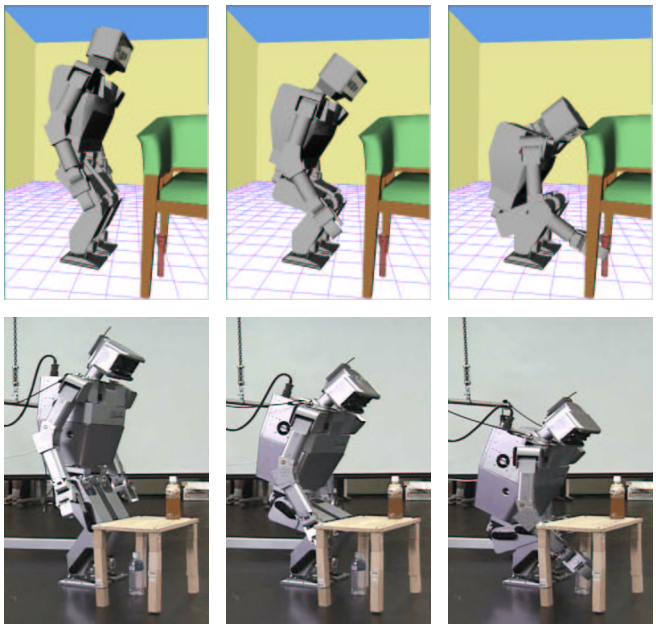

Figure 1: Dynamically-stable motion for retrieving an object (*top*: simulation, *bottom*: real robot hardware).

2 Background

Due to the complexity of motion planning in its general form [32], the use of complete algorithms [5, 12, 33] is limited to low-dimensional configuration spaces. This has motivated the use of heuristic algorithms, many of which employ randomization (e.g., [1, 2, 3, 4, 14, 15, 17,

20, 26]). Although these methods are incomplete, many have been shown to find paths in high-dimensional configuration spaces with high probability.

Motion planning for humanoid robots poses a particular challenge. Developing practical motion planning algorithms for humanoid robots is a daunting task given that humanoid robots typically have 30 or more degrees of freedom. The problem is further complicated by the fact that humanoid robots must be controlled very carefully in order to maintain overall static and dynamic stability. These constraints severely restrict the set of allowable configurations and prohibit the direct application of existing motion planning techniques. Although efficient methods have been developed for maintaining dynamic balance for biped robots [36, 31, 30, 16], none consider obstacle avoidance.

Motion planning algorithms that account for system dynamics typically approach the problem in one of two ways: 1) decoupling the problem by first computing a kinematic path, and subsequently transforming the path into a dynamic trajectory, or 2) searching the system state-space directly by reasoning about the possible controls that can be applied. The method presented in this paper adopts the first approach. Other methods using one of these two planning strategies have been developed for off-road vehicles[35, 6], free-flying 2D and 3D rigid bodies[23, 24], helicopters and satellites[8], and for a free-flying disc among moving obstacles[18]. None of these previous methods have yet been applied to complex articulated models such as humanoid robots. One notable exception is the VHRP simulation software under development[28], which contains a path planner that limits the active body degrees of freedom for humanoid robots for simultaneous obstacle avoidance and balance control. Since the space of possible computed motions is limited however, this planner is not fully general.

3 Dynamically-stable Motion Planning

Our approach is to adapt a variation of the randomized planner described in [20] to compute full-body motions for humanoid robots that are both dynamically-stable and collision-free. The first phase computes a statically-stable, collision-free path, and the second phase smooths and transforms this path into a dynamically-stable trajectory for the entire body. The planning method (RRT-Connect) and its variants utilize Rapidly-exploring Random Trees (RRTs) [22, 24] to connect two search trees, one from the initial configuration and the other from the goal. This method has been shown to be efficient in practice and converge towards a uniform exploration of the search space.

Robot Model and Assumptions: We have based our experiments on an approximate model of the H6 humanoid robot (see Figure 1), including the kinematics and dynamic properties of the links. Although we have focused our experiments on biped robots with a humanoid shape, the algorithm generally applies to any robot subject to balance constraints (legged or not). Aside from the existence of the dynamic model, we make the following assumptions:

1. **Environment model:** We assume that the robot has access to a 3D model of the surrounding environment to be used for collision checking.
2. **Initial posture:** The robot is currently balanced in a collision-free, statically-stable configuration supported by either one or both feet.
3. **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.
4. **Support base:** The location of the supporting foot (or *feet* in the case of dual-leg support) does not change during the planned motion.

3.1 Problem Formulation:

Our problem will be defined in a 3D world $\mathcal{W}$ in which the robot moves. $\mathcal{W}$ is modeled as the Euclidean space $\Re^3$ ($\Re$ is the set of real numbers).

Robot: Let the robot $\mathcal{A}$ be a finite collection of p rigid links $\mathcal{L}_i$ $(i = 1, \ldots, p)$ organized in a kinematic hierarchy with Cartesian frames $\mathcal{F}_i$ attached to each link. We denote the position of the center of mass c_i of link $\mathcal{L}_i$ relative to $\mathcal{F}_i$. A *pose* of the robot is denoted by the set $\mathcal{P} = \{T_1, T_2, \ldots, T_p\}$, of p relative transformations for each of the links $\mathcal{L}_i$ as defined by the frame $\mathcal{F}_i$ relative to its parent link's frame. The *base* or *root* link transformation T_1 is defined relative to some world Cartesian frame $\mathcal{F}_{world}$. Let n denote the number of generalized coordinates or *degrees of freedom* (DOFs) of $\mathcal{A}$. A *configuration* is denoted by $q \in \mathcal{C}$, a vector of n real numbers specifying values for each of the generalized coordinates of $\mathcal{A}$. Let $\mathcal{C}$ be the *configuration space* or $\mathcal{C}$-space of $\mathcal{A}$. $\mathcal{C}$ is a space of dimension n.

Obstacles: The set of obstacles in the environment $\mathcal{W}$ is denoted by $\mathcal{B}$, where $\mathcal{B}_k$ $(k = 1, 2, \ldots)$ represents an individual obstacle. We define the *$\mathcal{C}$-obstacle region* $\mathcal{C}B \subset \mathcal{C}$ as the set of all configurations $q \in \mathcal{C}$ where one or more of the links of $\mathcal{A}$ intersect (are in collision) with another link of $\mathcal{A}$, any of the obstacles $\mathcal{B}_k$. We also regard configurations $q \in \mathcal{C}$ where one or more *joint limits* are violated as part of the $\mathcal{C}$-obstacle region $\mathcal{C}B$. The open subset $\mathcal{C} \setminus \mathcal{C}B$ is denoted by $\mathcal{C}_{free}$ and its closure by $cl(\mathcal{C}_{free})$, and it represents the *space of collision-free configurations* in $\mathcal{C}$ of the robot $\mathcal{A}$.

Balance and Torque Constraints: Let $\mathcal{X}(q)$ be a vector relative to $\mathcal{F}_{world}$ representing the global position of the center of mass of $\mathcal{A}$ while in the configuration q. A configuration q is *statically-stable* if: 1) the projection of $\mathcal{X}(q)$ along the gravity vector g lies within the area of support $\mathcal{S}P$ (i.e. the convex hull of all points of contact

between $\mathcal{A}$ and the support surface in $\mathcal{W}$), and 2) the joint torques Γ needed to counteract the gravity-induced torques $G(q)$ do not exceed the maximum torque bounds Γ_{max}. Let $\mathcal{C}_{stable} \subset \mathcal{C}$ be the subset of *statically-stable configurations* of $\mathcal{A}$. Let $\mathcal{C}_{valid} = \mathcal{C}_{stable} \cap \mathcal{C}_{free}$ denote the subset of configurations that are both *collision-free* and *statically-stable* postures of the robot $\mathcal{A}$. $\mathcal{C}_{valid}$ is called the set of *valid* configurations.

Solution Trajectory: Let $\tau : \mathcal{I} \mapsto \mathcal{C}$ where $\mathcal{I}$ is an interval $[t_0, t_1]$, denote a motion trajectory or *path* for $\mathcal{A}$ expressed as a function of time. $\tau(t)$ represents the configuration q of $\mathcal{A}$ at time t, where $t \in \mathcal{I}$. A trajectory τ is said to be *collision-free* if $\tau(t) \in \mathcal{C}_{free}$ for all $t \in \mathcal{I}$. A trajectory τ is said to be both *collision-free* and *statically-stable* if $\tau(t) \in \mathcal{C}_{valid}$ for all $t \in \mathcal{I}$. Given $q_{init} \in \mathcal{C}_{valid}$ and $q_{init} \in \mathcal{C}_{valid}$, we wish to compute a continuous motion trajectory τ such that $\forall t \in [t_0, t_1]$, $\tau(t) \in \mathcal{C}_{valid}$, and $\tau(t_0) = q_{init}$ and $\tau(t_1) = q_{goal}$. We refer to such a trajectory as a *statically-stable* trajectory.

Dynamic Stability: Theoretically, any statically-stable trajectory can be transformed into a dynamically-stable trajectory by arbitrarily slowing down the motion. For these experiments, we utilize the online balance compensation scheme described in [16] as a method of generating a final dynamically-stable trajectory after path smoothing (see Section 3.4).

Planning Query: Note that in general, if a dynamically-stable solution trajectory exists for a given path planning query, there will be many such solution trajectories. Let Φ denote the set of all dynamically-stable solution trajectories for a given problem. The query $Planner(\mathcal{A}, \mathcal{B}, q_{init}, q_{goal}) \longrightarrow \tau$, accepts as input the robot and obstacle models along with the initial and goal postures, and attempts to calculate a solution trajectory $\tau \in \Phi$. If no solution is found, τ will be empty (a null trajectory).

3.2 Path Search

Unfortunately, there are no currently known methods for explicitly representing $\mathcal{C}_{valid}$. The obstacles are modeled completely in $\mathcal{W}$, thus an explicit representation of $\mathcal{C}_{free}$ is also not available. However, using a collision detection algorithm, a given $q \in \mathcal{C}$ can be tested to determine whether $q \in \mathcal{C}_{free}$. Testing whether $q \in \mathcal{C}_{stable}$ can also be checked verifying that the projection of $\mathcal{X}(q)$ along g is contained within the boundary of $\mathcal{SP}$, and that the torques Γ needed to counteract gravitational torques $G(q)$ do not exceed Γ_{max}.

Distance Metric: As with the most planning algorithms in high-dimensions, a metric ρ is defined on $\mathcal{C}$. The function $\rho(q, r)$ returns some measure of the distance between the pair of configurations q and r. Some axes in $\mathcal{C}$ are weighted relative to each other, but the general idea is to measure the "closeness" of pairs of configurations with a positive scalar function.

For our humanoid robot models, we employ a metric that assigns higher relative weights to the generalized coordinates of links with greater mass and proximity to the trunk (torso): $\rho(q, r) = \sum_{i=1}^{n} w_i ||q_i - r_i||$. This choice of metric function attempts to heuristically encode a general relative measure of how much the variation of an individual joint parameter affects the overall body posture. Additional experimentation is needed in order to evaluate the efficacy of the many different metric functions possible.

Planning Algorithm: We employ a randomized search strategy based on Rapidly-exploring Random Trees (RRTs) [23, 20]. For implementation details and analysis of RRTs, the reader is referred to the original papers or a summary in [24]. In [21], we developed an RRT variant the generates search trees using a dynamics filter function to guarantee dynamically-stable trajectories along each incremental search motion.

The algorithm described in this paper is more general and efficient than the planner presented in [21], since it does not require the use of a dynamics filtering function during the path search phase. In addition, it can handle either single or dual-leg support postures, and the calculated trajectories have been verified using real robot hardware. The basic idea is the same as the RRT-Connect algorithm described in [20]. The key difference is that *instead of searching $\mathcal{C}$ for a solution path that lies within $\mathcal{C}_{free}$, the search is performed in $\mathcal{C}_{stable}$ for a solution path that lies within $\mathcal{C}_{valid}$.* In particular, we modify the planner variant that employs symmetric calls to the *EXTEND* function as follows:

1. The *NEW_CONFIG* function in the *EXTEND* operation checks balance constraints in addition to checking for collisions with obstacles (i.e. $q_{new} \in \mathcal{C}_{valid}$).

2. Rather than picking a purely random configuration $q_{rand} \in \mathcal{C}$ at every planning iteration, we pick a random configuration that also happens to correspond to a statically-stable posture of the robot (i.e. $q_{rand} \in \mathcal{C}_{stable}$).

Pseudocode for the complete algorithm is given in Figure 2. The main planning loop involves performing a simple iteration in which each step attempts to extend the RRT by adding a new vertex that is biased by a randomly-generated, statically-stable configuration (see Section 3.3). *EXTEND* selects the nearest vertex already in the RRT to the given configuration, q, with respect to the distance metric ρ. Three situations can occur: *Reached*, in which q is directly added to the RRT, *Advanced*, in which a new vertex $q_{new} \neq q$ is added to the RRT; *Trapped*, in which no new vertex is added due to the inability of *NEW_CONFIG* to generate a path segment towards q that lies within $\mathcal{C}_{valid}$.

NEW_CONFIG attempts to make an incremental motion toward q. Specifically, it checks for the existence of a short path segment $\delta = (q_{near}, q_{new})$ that lies entirely within $\mathcal{C}_{valid}$. If $\rho(q, q_{near}) < \epsilon$, where ϵ is some fixed incremental distance, then q itself is used as the new configuration q_{new} at the end of the candidate path segment δ (i.e. $q_{new} = q$). Otherwise, q_{new} is generated at

```
EXTEND(T, q)
 1  q_near ← NEAREST_NEIGHBOR(q, T);
 2  if NEW_CONFIG(q, q_near, q_new) then
 3      T.add_vertex(q_new);
 4      T.add_edge(q_near, q_new);
 5      if q_new = q then Return Reached;
 6      else Return Advanced;
 7  Return Trapped;
```

```
RRT_CONNECT_STABLE(q_init, q_goal)
 1  T_a.init(q_init); T_b.init(q_goal);
 2  for k = 1 to K do
 3      q_rand ← RANDOM_STABLE_CONFIG();
 4      if not (EXTEND(T_a, q_rand) = Trapped) then
 5          if (EXTEND(T_b, q_new) = Reached) then
 6              Return PATH(T_a, T_b);
 7      SWAP(T_a, T_b);
 8  Return Failure
```

Figure 2: Pseudocode for the dynamically-stable motion planning algorithm.

a distance ϵ along the straight-line from q_{near} to q.[1] All configurations q' along the path segment δ are checked for collision, and tested whether balance constraints are satisfied. Specifically, if $\forall q' \in \delta(q_{near}, q_{new}), q' \in \mathcal{C}_{valid}$, then *NEW_CONFIG* succeeds, and q_{new} is added to the tree $\mathcal{T}$. In this way, the planner uses uniform samples of $\mathcal{C}_{stable}$ in order to grow trees that lie entirely within $\mathcal{C}_{valid}$.

Convergence and Completeness: Although not given here, arguments similar to those presented in [20] and [24] can be constructed to show uniform coverage and convergence over $\mathcal{C}_{valid}$.

Ideally, we would like to build a *complete* planning algorithm. That is, the planner always returns a solution trajectory if one exists, and indicates failure if no solution exists. As mentioned in Section 2, implementing a practical complete planner is a daunting task for even low-dimensional configuration spaces (see [13]). Thus, we typically trade off completeness for practical performance by adopting heuristics (e.g. randomization).

The planning algorithm implemented here is incomplete in that it returns failure after a preset time limit is exceeded. Thus, if the planner returns failure, we cannot conclude whether or not a solution exists for the given planning query, only that our planner was unable to find one in the allotted time. Uniform coverage and convergence proofs, though only theoretical, at least help to provide some measure of confidence that when an algorithm fails to find a solution, it is likely that no solution exists. This is an area of ongoing research.

[1] A slight modification must be made for the case of dual-leg support. In this case, when interpolating two stable configurations, inverse kinematics for the leg is used to force the relative position between the feet to remain fixed. The same s technique is also used for generating random statically-stable dual-leg postures (see Section 3.3).

3.3 Random Statically-stable Postures

For our algorithm to work, we require a method of generating random statically-stable postures (i.e. random point samples of $\mathcal{C}_{stable}$). Although it is trivial to generate random configurations in $\mathcal{C}$, it is not so easy to generate them in $\mathcal{C}_{stable}$, since it encompasses a much smaller subset of the configuration space.

In our current implementation, a set $\mathcal{Q}_{stable} \subset \mathcal{C}_{stable}$ of N samples of $\mathcal{C}_{stable}$ is generated as a preprocessing step. This computation is specific to a particular robot and support-leg configuration, and need only be performed once. Different collections of stable postures are saved to files and can be loaded into memory when the planner is initialized. Although stable configurations could be generated "on-the-fly" at the same time the planner performs the search, pre-calculating $\mathcal{Q}_{stable}$ is preferred for efficiency. In addition, multiple stable-configuration set files for a particular support-leg configuration can be saved independently. If the planner fails to find a path after all N samples have been removed from the currently active $\mathcal{Q}_{stable}$ set, a new one can be loaded with different samples.

Single-leg Support Configurations: For configurations that involve balancing on only one leg, the set $\mathcal{Q}_{stable}$ can be populated as follows:

1. The configuration space of the robot $\mathcal{C}$ is sampled by generating a random body configuration $q_{rand} \in \mathcal{C}$.

2. Assuming the right leg is the supporting foot, q_{rand} is tested for membership in $\mathcal{C}_{valid}$ (i.e. static stability, no self-collision, and joint torques below limits).

3. Using the same sample q_{rand}, a similar test is performed assuming the left leg is the supporting foot.

4. Since most humanoid robots have left-right symmetry, if $q_{rand} \in \mathcal{C}_{valid}$ in either or both cases, we can "mirror" q_{rand} to generate stable postures for the opposite foot.

Dual-leg Support Configurations: It is slightly more complicated to generate statically-stable body configurations supported by both feet at a given fixed relative position. In this case, populating $\mathcal{Q}_{stable}$ is very similar to the problem of sampling the configuration space of a constrained closed-chain system (e.g. closed-chain manipulator robots or molecular conformations [25, 10]). The set $\mathcal{Q}_{stable}$ is populated with fixed-position dual-leg support postures as follows:

1. As in the single-leg case, the configuration space of the robot $\mathcal{C}$ is sampled by generating a random body configuration $q_{rand} \in \mathcal{C}$.

2. Holding the right leg fixed at its random configuration, inverse kinematics is used to attempt to position the left foot at the required relative position to generate the body configuration q_{right}. If it succeeds, then q_{right} is tested for membership in $\mathcal{C}_{valid}$.

3. An identical procedure is performed to generate q_{left} by holding the left leg fixed at its random configuration derived from q_{rand}, using inverse kinematics to position the right leg, and testing for membership in $\mathcal{C}_{valid}$.

4. If either $q_{right} \in \mathcal{C}_{valid}$ or $q_{left} \in \mathcal{C}_{valid}$, and the robot has left-right symmetry, additional stable postures can be derived by mirroring the generated stable configurations.

3.4 Trajectory Generation

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:

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 in $\mathcal{C}_{valid}$.[2] This step typically eliminates any potentially unnatural postures along the random path (e.g. unnecessarily large arm motions). The resulting smoothed path is transformed into an input trajectory using a minimum-jerk model [7].

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 [16], which enforces constraints upon the zero moment point (ZMP) trajectory in order to maintain overall dynamic stability. The output configuration of the filter is guaranteed to lie in $\mathcal{C}_{stable}$. Collision-checking is used to verify that the final output trajectory lies in $\mathcal{C}_{valid}$, with the motion made slower in the case of collision.
Although this method has generated satisfactory results in our experiments, it is by no means the only option. Other ways of generating dynamically-stable trajectories from a given input motion are also potentially possible to apply here (e.g. [37, 29]). It is also possible to employ variational techniques, or apply algorithms for computing time-optimal trajectories [34]. Calculating the *globally-optimal trajectory* according to some cost functional based on the obstacles and the dynamic model is an open problem, and an area of ongoing research.

[2]When interpolating dual-leg configurations, inverse kinematics is used to keep the relative position of the feet fixed.

Task Description	Computation Time (seconds)			
	min	max	avg	stdev
Reach under chair	171	598	324	138
Lift leg over box	26	103	48	21
Reach over table	194	652	371	146

Table 1: Performance statistics ($N = 25$ trials).

4 Experiments

This section presents some preliminary experiments performed on a 270 MHz SGI O2 (R12000) workstation. We have implemented a prototype planner in C++ that runs within a graphical simulation environment [19]. 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 on an actual humanoid robot hardware platform. The "H6" humanoid robot (33-DOF) is 137cm tall and weighs 51kg (including 4kg of batteries). Figure 1 shows a computed dynamically-stable motion for the H6 robot moving from a neutral standing position to a low crouching position in order to retrieve an object from beneath a chair. Figure 3 shows a different view of the real robot executing the same motion. Figure 4 shows a motion for positioning the right leg above the top of a box while balancing on the left leg. The motion in Figure 5 was not executed on the real robot, but is interesting in that it involves reaching for an object placed on top of a cabinet while avoiding both the cabinet and the shelves behind the robot. The robot is required to balance on one leg in order to extend the arm far enough to reach the obstacle on the table.

Each of the scenes contains over 9,000 triangle primitives. The 3D collision checking software used for these experiments was the RAPID library based on OBB-Trees developed by the University of North Carolina[9]. The total wall time elapsed in solving these queries ranges from under 30 seconds to approximately 11 minutes. A summary of the computation times for repeated runs of 25 trials each is shown in Table 1.

5 Discussion

This paper presents an algorithm for computing dynamically-stable collision-free trajectories given full-body posture goals. Although we have focused our experiments on biped robots with a humanoid shape, the algorithm is general and can be applied to any robot subject to balance constraints (legged or not). There are many potential uses for such software, with the primary one being a high-level control interface for automatically computing motions to solve complex tasks for humanoid

robots that involve simultaneous obstacle-avoidance and balance constraints.

The limitations of the algorithm form the basis for our future work: 1) the current implementation of the planner can only handle a fixed position for either one or both feet. 2) The effectiveness of different configuration space distance metrics needs to be investigated. 3) We currently have no method for integrating visual or tactile feedback.

Acknowledgments: We thank Fumio Kanehiro and Yukiharu Tamiya for their efforts in developing the AutoBalancer software library. We are grateful to Steven LaValle and Hirohisa Hirukawa for helpful discussions. This research is supported in part by a Japan Society for the Promotion of Science (JSPS) Postdoctoral Fellowship for Foreign Scholars in Science and Engineering, and by JSPS Grant-in-Aid for Research for the Future (JSPS-RFTF96P00801).

References

[1] N. Amato and Y. Wu. A randomized roadmap method for path and manipuation planning. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, pages 113–120, 1996.

[2] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, December 1990.

[3] R. Bohlin and L. Kavraki. Path planning using Lazy PRM. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, April 2000.

[4] V. Boor, M. Overmars, and A.F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, 1999.

[5] J.F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[6] M. Cherif and C. Laugier. Motion planning of autonomous off-road vehicles under physical interaction constraints. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, May 1995.

[7] T. Flash and N. Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *J. Neurosci.*, 5(7):1688–1703, 1985.

[8] E. Frazzoli, M.A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicles motion planning. Technical report, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1999. Technical report LIDS-P-2468.

[9] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *SIGGRAPH '96 Proc.*, 1996.

[10] Li Han and Nancy M. Amato. A kinematics-based probabilistic roadmap method for closed chain systems. In *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, March 2000.

[11] Kazuo Hirai. Current and future perspective of honda humanoid robot. In *Proc. IEEE/RSJ Int. Conf. Intell. Robot. & Sys. (IROS)*, pages 500–508, 1997.

[12] H. Hirukawa, B. Mourrain, and Y. Papegay. A symbolic-numeric silhouette algorithm. In *Proc. IEEE/RSJ Int. Conf. Intell. Robot. & Sys. (IROS)*, November 2001.

[13] H. Hirukawa and Y. Papegay. Motion planning of objects in contact by the silhouette algorithm. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, pages 722–729, April 2000.

[14] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom : Random reflections at c-space obstacles. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, pages 3318–3323, April 1994.

[15] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. Comput. Geom. & Appl.*, 9(4-5):495–512, 1997.

[16] S. Kagami, F. Kanehiro, Y. Tamiya, M. Inaba, and H. Inoue. AutoBalancer: An Online Dynamic Balance Compensation Scheme for Humanoid Robots. In *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, 2000.

[17] L. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Trans. Robot. & Autom.*, 12(4):566–580, 1996.

[18] R. Kindel, D. Hsu, J.C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, April 2000.

[19] J.J. Kuffner, S. Kagami, M. Inaba, and H. Inoue. Graphical simulation and high-level control of humanoid robots. In *Proc. IEEE/RSJ Int. Conf. Intell. Robot. & Sys. (IROS)*, November 2000.

[20] J.J. Kuffner and S.M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, April 2000.

[21] J.J. Kuffner, S.Kagami, M. Inaba, and H. Inoue. Dynamically- stable motion planning for humanoid robots. In *IEEE-RAS Int. Conf. Human. Robot. (Humanoids)*, Boston, MA, September 2000.

[22] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State Univ. `<http://janowiec.cs.iastate.edu/papers/rrt.ps>`, Oct. 1998.

[23] S.M. LaValle and J.J Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, May 1999.

[24] S.M. LaValle and J.J Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, March 2000.

[25] S.M. LaValle, J.H. Yakey, and L.E. Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, 1999.

[26] E. Mazer, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. Artificial Intell. Res.*, 9:295–316, November 1998.

[27] K. Nagasaka, M. Inaba, and H. Inoue. Walking pattern generation for a humanoid robot based on optimal gradient method. In *Proc. IEEE Int. Conf. Sys. Man. & Cyber.*, 1999.

[28] Y. Nakamura and *et. al.* V-HRP: Virtual humanoid robot platform. In *IEEE-RAS Int. Conf. Human. Robot. (Humanoids)*, September 2000.

[29] Y. Nakamura and K. Yamane. Interactive motion generation of humanoid robots via dynamics filter. In *Proc. of First IEEE-RAS Int. Conf. on Humanoid Robots*, September 2000.

[30] J. Pratt and G. Pratt. Exploiting natural dynamics in the control of a 3d bipedal walking simulation. In *In Proc. of Int. Conf. on Climbing and Walking Robots (CLAWAR99)*, September 1999.

[31] Marc Raibert. *Legged Robots that Balance*. MIT Press, Cambridge, MA, 1986.

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

[33] J. T. Schwartz and M. Sharir. On the 'piano movers' problem: Ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in applied Mathematics*, 4:298–351, 1983.

[34] Z. Shiller and S. Dubowsky. On computing time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Trans. Robot. & Autom.*, 7(7), December 1991.

[35] Z. Shiller and R.Y. Gwo. Dynamic motion planning of autonomous vehicles. *IEEE Trans. Robot. & Autom.*, 7(2):241–249, April 1991.

[36] M. Vukobratovic, B. Borovac, D. Surla, and D. Stokie. *Biped Locomotion: Dynamics, Stability, Control, and Applications*. Springer-Verlag, Berlin, 1990.

[37] J. Yamaguchi, S. Inoue, D. Nishino, and A. Takanishi. Development of a bipedal humanoid robot having antagonistic driven joints and three dof trunk. In *Proc. IEEE/RSJ Int. Conf. Intell. Robot. & Sys. (IROS)*, pages 96–101, 1998.

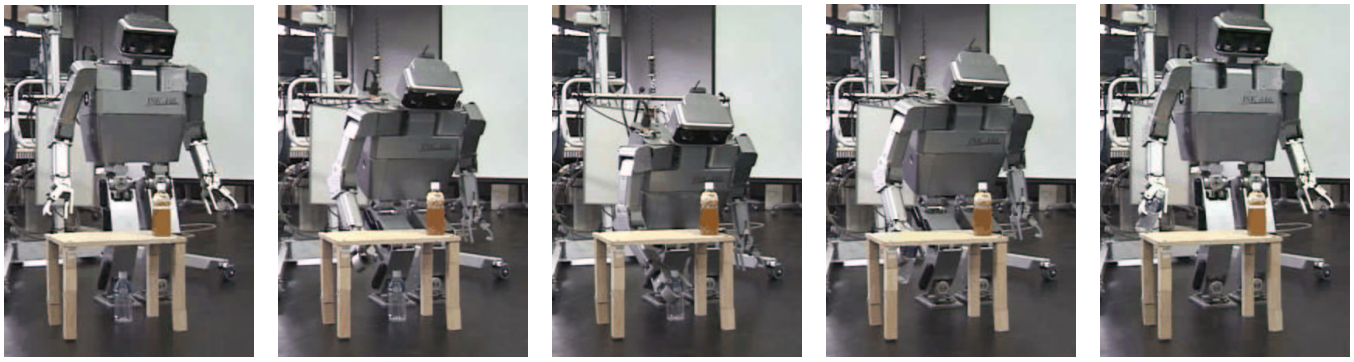

Figure 3: Dynamically-stable crouching trajectory for retrieving an object from beneath an obstacle
.

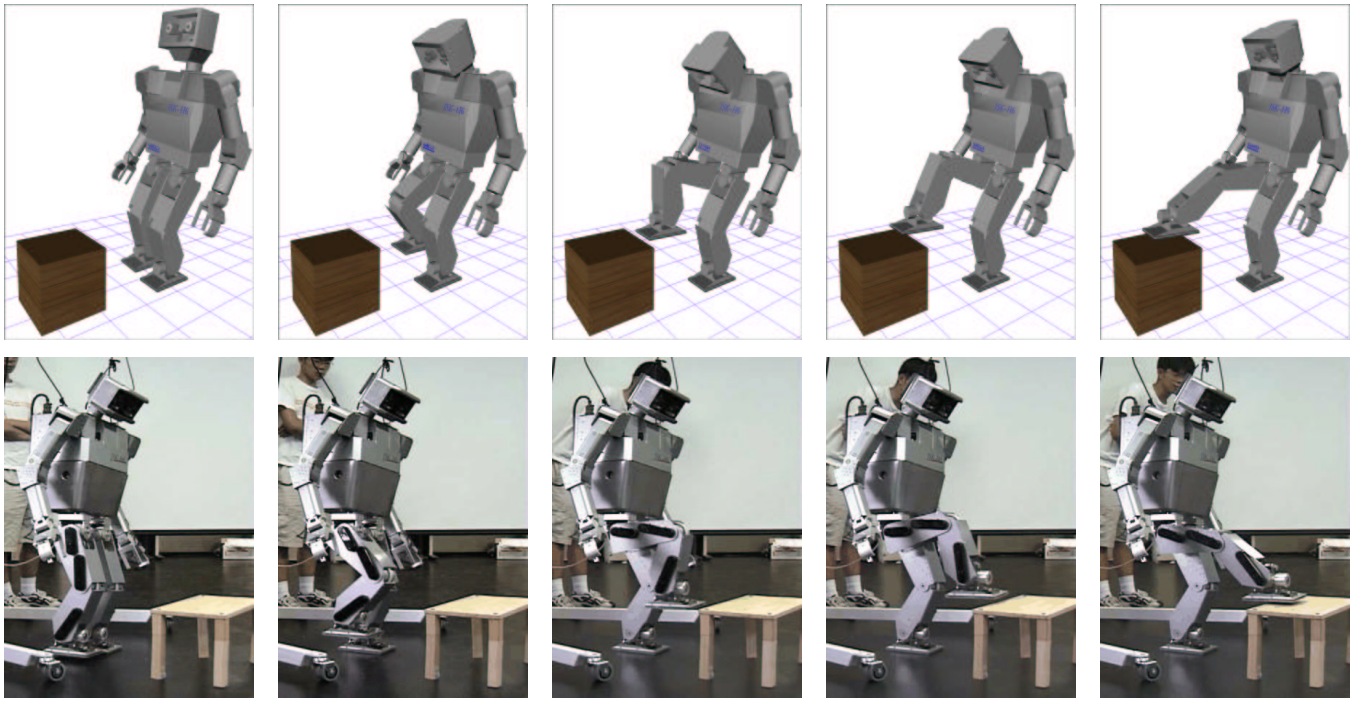

Figure 4: Positioning the right foot above an obstacle while balancing on the left leg. (*top*: simulation, *bottom*: actual hardware).

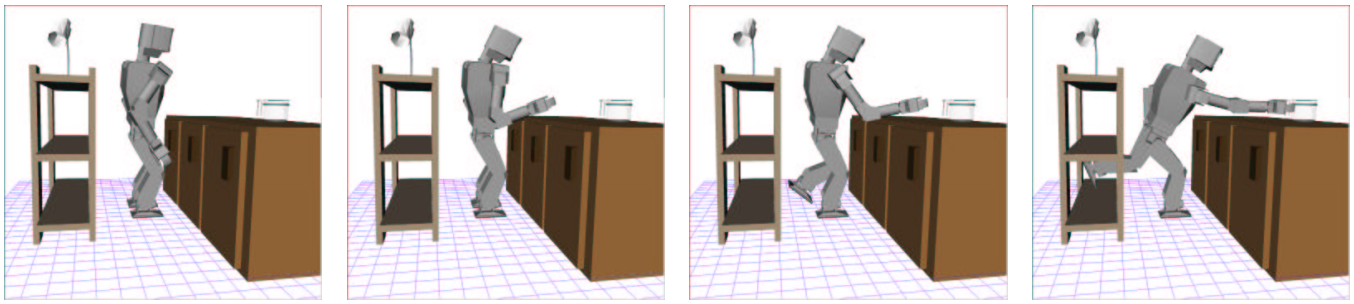

Figure 5: Reaching for an object atop a cabinet while avoiding obstacles and balancing on the right leg.