In Proc. 11th Int'l Symp. of Robotics Research (ISRR 2003)

Motion Planning for Humanoid Robots

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

[1] The Robotics Institute, Carnegie Mellon University
kuffner@cs.cmu.edu
http://www.kuffner.org/james
[2] Digital Human Research Center
National Institute of Advanced Industrial Science and Technology (AIST)
{s.kagami,j.kuffner}@aist.go.jp
http://www.dh.aist.go.jp/
[3] School of Information Science and Technology, The University of Tokyo
{nishi,inaba,inoue}@jsk.t.u-tokyo.ac.jp
http://www.jsk.t.u-tokyo.ac.jp/

Abstract. Humanoid robotics hardware and control techniques have advanced rapidly during the last five years. Presently, several companies have announced the commercial availability of various humanoid robot prototypes. In order to improve the autonomy and overall functionality of these robots, reliable sensors, safety mechanisms, and general integrated software tools and techniques are needed. We believe that the development of practical motion planning algorithms and obstacle avoidance software for humanoid robots represents an important enabling technology. This paper gives an overview of some of our recent efforts to develop motion planning methods for humanoid robots for application tasks involving navigation, object grasping and manipulation, footstep placement, and dynamically-stable full-body motions. We show experimental results obtained by implementations running within a simulation environment as well as on actual humanoid robot hardware.

1 Introduction

Humanoid robotics technology has recently made rapid progress, and the commercial availability of humanoid robot hardware will happen very soon. This will lead to a rising demand for software and algorithms useful to improving the usability and autonomy of humanoids. One important area of need will be software for safe, autonomous operation in human environments such as homes and offices. Thus, the development of general-purpose motion planning tools and methods for humanoid robots represents a very challenging research area.

This paper presents 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. In the latter case, we consider the problem of computing dynamically-stable, collision-free trajectories for the entire 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.

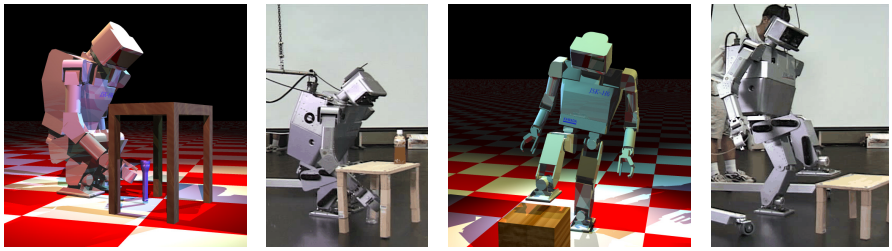

Fig. 1. Simulation and video snapshots of planned full-body trajectories.

2 Footstep Planning

Global path planning and obstacle avoidance strategies for mobile robots and manipulators has a large and extensive history in the robotics literature (e.g. see [15,8] 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 2D environment. Because of 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 2D 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.

Our approach is to build a search tree from a discrete set of feasible footstep locations corresponding to available stepping motions. Using standard dynamic programming techniques, we compute a sequence of footstep placements to reach a goal region in an obstacle-cluttered environment. Optimal sequences of footstep placements can be computed according to the encoded heuristics that minimize the number and complexity of the steps taken. Such a strategy can be computed efficiently on standard PC hardware (under one second for simple environments, and in a few seconds for relatively complex, cluttered environments).

Biped Navigation Model: The biped model comes with a pre-determined set of feasible footstep locations for each foot. For example, Figure 3 shows the continuous, feasible footstep range FR_{right} for the right foot while supported by the left foot, and an example discrete 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 maneuver 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,

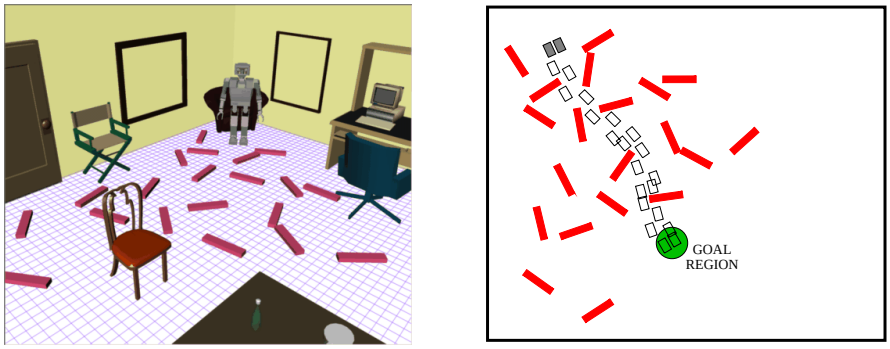

Fig. 2. *left:* Humanoid navigating in a cluttered office; *right:* planned footstep locations (Top view).

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 [12], or generated using an online algorithm [5].

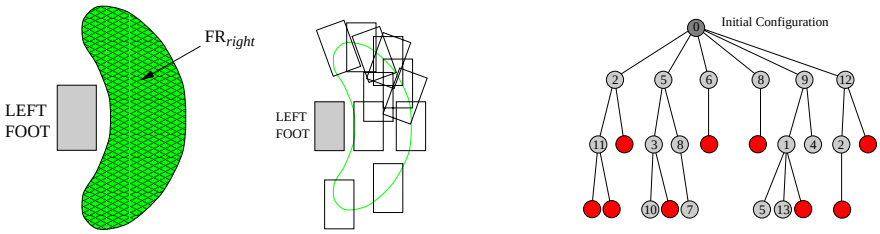

Fig. 3. Reachable placement positions for the right foot: (*left*: continuous region, and *middle*: discrete placements. *right*: search tree with pruned successor states (*red*) that resulted in bad foot placements or collisions.

Footstep Planning Algorithm: The planner accepts as input a discrete set of robot footprint locations, a trajectory generator, and a heuristic cost function. Both 2D and 3D representations of the robot and environment model can be used for collision checking (see [5]). 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 an 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:

$$\mathcal{L}(Q) = w_d D(N_Q) + w_\rho \rho(N_Q) + w_g \mathcal{X}(Q, Q_g)$$

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. $\rho(N_Q)$ is a function that encodes the path "goodness", such as favoring "safe" overall foot placements, as well as paths which incur few orientation changes (for detailed example path metrics, see [5]). These terms have the combined effect of favoring paths with fewer steps, as well as slightly favoring paths with long sequences of straight-line steps. The final term represents an estimated cost from the current configuration to the goal region. $\mathcal{X}(Q, Q_g)$ approximates the minimum number of steps needed to traverse the straight-line distance between the footprint location at Q, and a footprint in the center of the goal region Q_g. Each of the terms are weighted relative to each other by the factors w_d, w_ρ, and w_g.

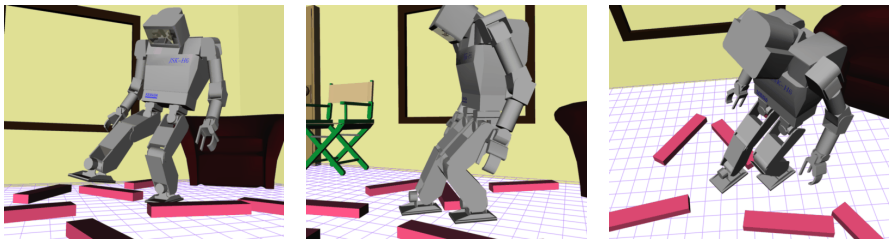

Fig. 4. Simulation snapshots during execution of footstep plan.

Figure 2 shows a cluttered office in which a model of the Humanoid robot "H6" must navigate, and a top view of a footstep sequence computed to reach a circular goal region in the center 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 seconds on an 1.6 GHz Pentium4 running Linux. We used a 2D polygon-polygon intersection test for the first phase of collision-checking, and the (*V-clip*) library (see [18]) for fast minimum distance determination between the obstacles and the convex hull of each leg link for the second phase.

3 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.

Due to the complexity of motion planning in its general form [19], the use of complete algorithms limited to low-dimensional configuration spaces. Even single-arm manipulation planning (typically 6-7 DOFs) 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 development of numerous planning methods, many of which employ techniques such as randomization (e.g. [2,10,7,17,11]), lazy evaluation of collision checking (e.g. [3,20]), deterministic sampling [4], 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.

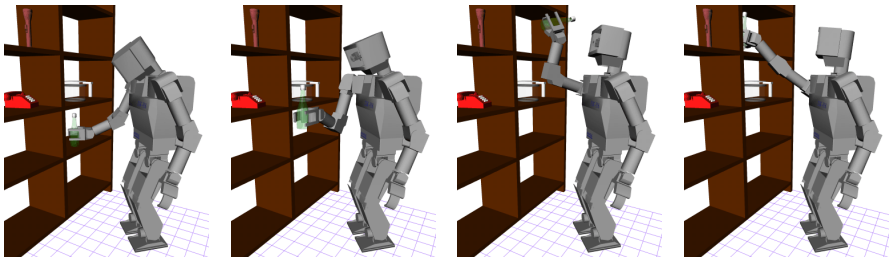

Fig. 5. Manipulation planning for the 'H6' robot.

We have adopted an efficient general path planning algorithm that is well-suited for manipulation planning. The algorithm, *RRT-Connect* [11], was originally developed to plan collision-free motions for animated characters in 3D virtual environments[14]. It uses a randomized search strategy based on Rapidly-exploring Random Trees (RRTs) [16]. Distinguishing features of this algorithm include no preprocessing of the workspace (ideal for changing environments), greedy behavior that solves simple queries very efficiently, and uniform coverage of any non-convex space (For details and analysis, see [11]). This planner has been used to efficiently solve several well-known motion planning benchmarks, such as the "Alpha 1.0 puzzle" and "Flange 1.0 problem" (Figure 6), which are considered difficult due to the presence of narrow passages in the configuration space. These examples have been analyzed and made publicly available for research purposes[1]. Computation times for our

planner on the Alpha 1.0 puzzle range from 12 minutes to 3.5 hours on a 900 MHz PC running Linux (without any parameter tuning). Many previous planners have been unable to solve this problem without assistance.

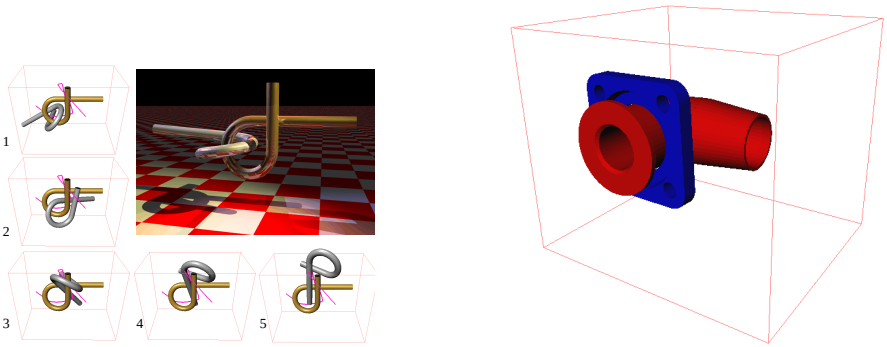

Fig. 6. *left:* The "Alpha Puzzle" Benchmark. *right:* The "Flange" Benchmark.

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 5 shows snapshots of a planned motion for the 'H6' humanoid repositioning a bottle from the lower shelf to the upper shelf. In the examples shown in Figure 7, 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 behavior in an attempt to find the missing object.

Fig. 7. *left:* Grasping a coffee pot. *right:* Answering the telephone.

4 Full-body Motions

Automatic, full-body motion planning for humanoid robots presents a formidable computational challenge due to: 1) the high number of degrees of freedom, 2)

complex kinematic and dynamic models, and 3) 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[13]. Obstacle and balance constraints are imposed upon incremental search motions. Provided 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 and dynamically-stable trajectory for the entire body.

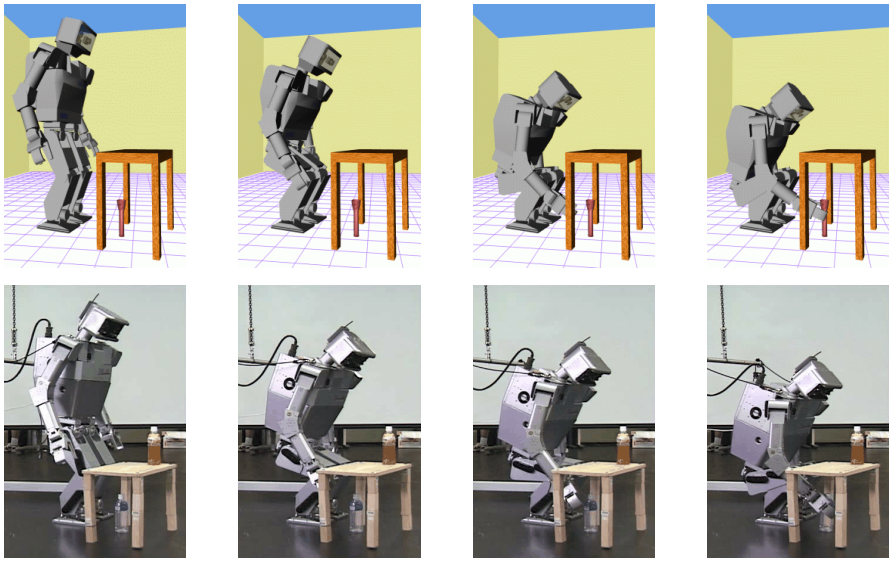

Fig. 8. Dynamically-stable motion for retrieving an object (*top*: simulation, *bottom*: actual hardware).

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:

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.

Full-body Trajectory Generation: The key idea of the planning algorithm is to search the space of statically-stable configurations ($\mathcal{C}_{stable}$) for a solution path that also lies within the free configuration space ($\mathcal{C}_{free}$). Each incremental search motion checks balance constraints while also checking for collisions with obstacles. Rather than picking a purely random configurations as a target for every planning iteration, we pick from a pre-generated set of statically-stable postures (i.e. $q_{rand} \in \mathcal{C}_{stable}$). For a more detailed explanation, see [13].

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:

1. **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 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 [6].
2. **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 [9], 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 $\mathcal{C}_{free}$, with the motion made slower in the case of collision.

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 on an actual humanoid robot hardware platform. The "H6" humanoid robot (33-DOF) is 137cm tall and weighs 51kg (including 4kg of batteries). Figure 8 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 9 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 9,000 triangle primitives. The total wall time elapsed in solving these queries ranges from under 5 seconds to approximately 1.5 minutes on a 900 MHz Pentium III running Linux.

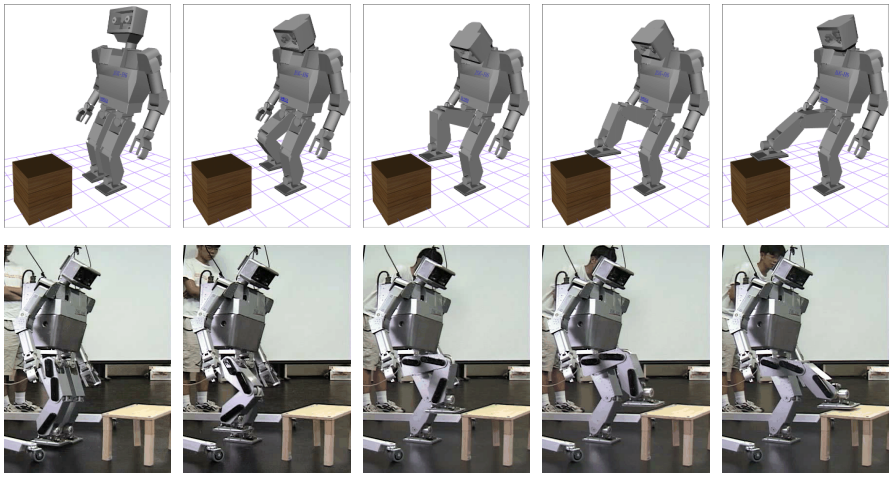

Fig. 9. Placing the right foot above the surface of an obstacle while balancing on the left leg. (*top*: simulation, *bottom*: actual hardware).

5 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 motion generation tools and techniques for humanoid robots represents a very challenging research area. We have given a brief overview of some our efforts to develop practical motion planning software for humanoids performing a variety of tasks. By 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. We believe that through the use of such kinds of task-level planning algorithms and interactive simulation software, the current and future capabilities of humanoid and other complex robotic systems can be improved.

Acknowledgments

We thank Fumio Kanehiro and Yukiharu Tamiya for their efforts in developing the Auto-Balancer software library. This research was 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), and NSF grants ECS-0325383, ECS-0326095, and ANI-0224419.

References

1. N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Trans. Robot. & Autom.*, 16(4):442–447, August 2000.

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. Michael S. Branicky, Steven M. LaValle, Kari Olson, and Libo Yang. Quasi-randomized path planning. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, 2001.
5. J. Chestnutt, J.J. Kuffner, K.Nishiwaki, and S.Kagami. Planning biped navigation strategies in complex environments. In *Proc. IEEE Int. Conf. on Humanoid Robotics (Humanoids'03)*, 2003.
6. T. Flash and N. Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *J. Neurosci.*, 5(7):1688–1703, 1985.
7. 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.
8. Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Trans. Robot. & Autom.*, 8(1):23–32, February 1992.
9. 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.
10. 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.
11. 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.
12. J.J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep planning among obstacles for biped robots. In *Proc. IEEE/RSJ Int. Conf. Intell. Robot. & Sys. (IROS)*, October 2001.
13. J.J. Kuffner, S.Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. Dynamically-stable motion planning for humanoid robots. *Autonomous Robots (special issue on Humanoid Robotics)*, 12(1):105–118, 2002.
14. J.J. Kuffner Jr. *Autonomous Agents for Real-time Animation*. PhD thesis, Stanford University, Stanford, CA, December 1999.
15. J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
16. S.M. LaValle and J.J Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int. Conf. Robot. & Autom. (ICRA)*, May 1999.
17. E. Mazer, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. Artificial Intell. Res.*, 9:295–316, November 1998.
18. B. Mirtich. VClip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.
19. 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.
20. G. Sanchez and J.C. Latombe. On delaying collision checking in prm planning – application to multi-robot coordination. *Int. J. Robot. Res.*, 21(1):5–26, January 2002.