Path Planning based on Accelerated Simulated Annealing of an Artificial Potential Field

STAVROS VOUGIOUKAS
Department of Applied Informatics
University of Macedonia
Egnatia 156, 54006 Thessaloniki
GREECE
bougis@uom.gr

Abstract: - Our objective is to develop a general–purpose path planning system for holonomic kinematic devices, such as robotic manipulators, amidst stationary obstacles. The requirements for the path planner are that it should be able to generate paths for robots with many degrees of freedom (DOF ≥ 6) which operate in three–dimensions, to solve path planning problems for multiple cooperating robots, to plan paths in environments of non–trivial geometric complexity (number of geometric primitives in the workspace) , and that it should be reasonably fast, and its performance should deteriorate gracefully as a function of the problem complexity. A category of path planners operates by precomputing the robot's configuration space and then using some search technique to find the goal. Such algorithms are intractable for more than 3-4 DOF's, because the volume of the configuration space - and hence the search space - grows exponentially in the degrees of freedom. In another approach [1, 2] the planner incrementally builds a graph connecting the local minima of the potential function and concurrently searches through the graph. The search approach is inspired by global optimization techniques and uses a heuristic combination of gradient descent and random motion in order to guide the robot towards the goal and escape from local minima. This approach avoids precomputing the configuration space and has been reported to successfully handle high–dimensional problems.

In this work, path planning is treated as a global optimization problem, where the non-convex cost function is an artificial potential field constructed in the 3D workspace, and the obstacles represent the constraints. The method of simulated annealing (SA) [10] is used, to cause the robot to escape from local minima and converge towards the global minimum, i.e., the goal configuration. Standard SA may require tens of thousands of iterations, each one requiring a computationally expensive collision detection. For this reason, an accelerated algorithm is developed, which combines stochastic gradient descent optimization, together with the random motion of SA. The location of all local minima discovered during the search is also kept in memory, together with an estimate of their attraction area, so that the same ones are not revisited by the robot. The accelerated algorithm is probabilistically complete, i.e., it will find a goal, if one exists, with probability 1. Simulations involving the coordinated motion of 3 planar robots (9 DOF's) were successfully performed. The path planner was also integrated with a general purpose kinematic device simulator, in order to be used for geometrically complex, real–world problems.

Key-Words: - Robotics, path planning, potential fields, simulated annealing

1 Introduction

Path planning algorithms are classified as either *complete* or *approximate*. Complete planners are aimed at guaranteeing a solution, if there is one, or proving that there is no solution. Their complexity has been found to be exponential in the number of degrees of freedom, and polynomial in the number of obstacles [12, 4]. Hence, complete planners have been implemented only for specific problems, usually under restrictive simplifying assumptions [11, 6].

A category of planners operates by precomputing the robot's configuration space and then using some search technique to find the goal. This category includes the so called "cell decomposition" planners [6], and also the planners that try to compute a local–minimum free global potential function [8, 5]. However, since the volume of the configuration space grows exponentially in the degrees of freedom, such algorithms are intractable for more than 3-4 DOF's.

In another approach [1, 2] the planner incrementally builds a graph connecting the local minima of the potential function and concurrently searches through the graph. The search approach is inspired by global optimization techniques and uses a heuristic combination of gradient descent and random motion in order to guide the robot towards the goal and escape from local minima. This approach avoids precomputing the configuration space and has been reported to successfully handle high – dimensional problems.

In this paper we will present a methodology which uses an novel *accelerated simulated annealing* algorithm as a search mechanism for finding the global minimum of a potential function defined over the robot configuration space. Simulated annealing has been successfully used to provide optimal,or near–optimal solutions to difficult NP-hard problems, like the traveling salesman problem. Our approach does not involve the precomputation of the robot's configuration space, or any global adjacency graph, and therefore it is applicable to robots with many DOF's without unreasonable memory requirements. Furthermore, random techniques, like SA, seem to satisfy "better" the graceful–degradation requirement, because harder problems will tend to require longer running times. This is in contrast to, say, heuristic deterministic techniques, which are good for specific classes of problems (in which the heuristics apply), but often fail to solve even simple problems, which lie outside these classes.

2 Simulated Annealing

Simulated annealing (SA) is a *stochastic global optimization* algorithm, i.e., it tries to find the global optimum of an N dimensional energy function. In contrast with local optimization methods, SA moves both up and downhill the energy landscape, and as the optimization process proceeds, it focuses on the most promising area. The way it does so, can be briefly described as follows : given the current point $\mathbf{q}_0$, the SA makes a trial–move by *randomly* choosing a feasible trial point $\mathbf{q}$, i.e., a trial point that satisfies all the constraints. This point is typically generated using a Gaussian distribution

$$g(\Delta\mathbf{q}) = \frac{1}{(2\pi T)^{D/2}} e^{-\frac{\Delta\mathbf{q}^t \Delta\mathbf{q}}{2T}} \tag{1}$$

where $\Delta\mathbf{q} = \mathbf{q} - \mathbf{q}_0$. A trial point with lower energy than that of the current point corresponds to a downhill move, and is always accepted. Uphill moves may also be accepted, depending probabilistically on the size of the uphill move and on another parameter T, which is referred to as the *temperature*. The higher T and the smaller the size of the uphill move are, the more likely that move will be accepted. The acceptance probability is the Boltzmann distribution

$$h(\Delta E) = \frac{e^{-E_{k+1}/T}}{e^{-E_{k+1}/T} + e^{-E_k/T}} = \frac{1}{1 + e^{\Delta E/T}} \tag{2}$$

where ΔE represents the "energy" difference between the present and previous value of the energy (cost function), .i.e, $\Delta E = E_{k+1} - E_k$. If the trial is accepted, the algorithm moves on from that point. If it is rejected, another point is chosen instead for a trial evaluation.

Initially the temperature is high, and the SA samples different regions of the energy landscape.

As the process continues, the temperature is lowered and the random steps become smaller, and thus only small downhill moves are accepted. In the limit, when the temperature goes to zero, SA becomes essentially a gradient–descent method. Given $g(\Delta\mathbf{q})$, if the temperature is lowered not faster than

$$T(k) = \frac{T_0}{lnk} \tag{3}$$

it has been formally shown [10] that SA converges to the global minimum with probability 1. A more heuristic proof is given in [7]. In order to statistically assure, i.e., requiring many trials, that any point in $\mathbf{q}$–space can be sampled infinitely often in annealing–time (IOT), it suffices to prove that the products of probabilities of not generating a state $\mathbf{q}$ IOT for all annealing–times successive to some k_0 yield zero,

$$\prod_{k=k_0}^{\infty} (1 - g_k) = 0. \tag{4}$$

This is equivalent to

$$\sum_{k=k_0}^{\infty} g_k = \infty \tag{5}$$

But since $T(k)$ is given by eq. 3, eq. 1 gives

$$\sum_{k=k_0}^{\infty} g_k \geq \sum_{k=k_0}^{\infty} e^{-lnk} = \sum_{k=k_0}^{\infty} \frac{1}{k} = \infty \tag{6}$$

3 Using SA for Path Planning (SAPP)

In the context of path planning, SA can be used to minimize globally artificial potential fields. The initial robot configuration $\mathbf{q}_o$ corresponds to the initial point in the optimization procedure. SA produces a sequence of points $\{\mathbf{q}_o, \mathbf{q}_1 \cdots \mathbf{q}_g\}$, with the goal configuration $\mathbf{q}_g$ being the last point. This sequence of points constitutes a connected collision–free path only if each pair of consecutive points defines also a connected collision–free path. In the work of Barraquand and Latombe [3, 1] a Brownian motion implemented as a discrete random walk was used to create such paths. In our approach we use a variation of this method. When the current state is $\mathbf{q}$, instead of generating a trial configuration using a Gaussian temperature – dependent distribution centered at $\mathbf{q}$ (see eq. 1), SA performs a random walk of temperature–dependent duration, which has been shown [9] to result in a trial configuration that follows the Gaussian distribution centered at the current state. Once the trial state has been generated in this way, the rest of the SA algorithm can be applied exactly as it was described previously.

Hence, our SA based Path Planner (SAPP) can be described as follows. Given a current state and temperature parameter T, a trial state is generated by a random walk. The duration of the random walk depends on the T; the higher T, the longer the walk. If the energy (potential) of the trial state is lower than that of the current state, it is always accepted. Else, the Boltzmann probability distribution is used to decide whether this trial state will be accepted. If the trial state is accepted, it becomes the current state, and the SAPP continues from it. If it is rejected, another trial state keeps being generated (via random walks), until one is accepted. The temperature is decreased every time a trial state is generated, according to the logarithmic schedule given in eq. 3.

The computed path, in general, will not be smooth, because of the random walks. Thus, it is necessary to post-process it to obtain smooth motion. The particular smoothing algorithm we use will be described in Section 5.

The SAPP is evidently *probabilistic complete*, i.e., the probability to find a path, if one exists, tends towards 1 when the computation time tends towards infinity. This property stems directly from the SA convergence properties, which dictate that, for a Gaussian state generation distribution, and a logarithmic temperature schedule the SA converges to the global minimum with a probability 1.

Probabilistic path planners, such as SAPP have been shown [3] to be able to handle high – dimensional problems, something that existing deterministic planners cannot do. Path planning for *multiple robots* can be performed by simply appending all the configuration vectors of the individual robots together, and minimizing the potential function with respect to this "compound"

configuration (also called the state). Furthermore, the SA can perform optimization with *arbitrary constraints* imposed on the state. For example, we may wish to plan the coordinated motion of two mobile robots, requiring that their relative distance is less than 10 meters, because of distance–related communication constraints. Constraint satisfaction is guaranteed for the entire path, because every time a random state is generated, SA checks to see if it is feasible, i.e., if it satisfies all the constraints.

4 Accelerated SAPP (ASAPP)

The major shortcoming of the SAPP is speed. SA can be quite time–consuming, especially when using the standard Boltzmann state–generating distribution. Exponential or hyper-exponential speedup of the SA is possible by using a different distribution [13, 7]. Some type of random motion, in which steps are not independent of each other, might result in non–Gaussian distributions, but we are not aware of existing literature in this area; this could constitute a topic for future research.

Another reason why SAPP may take a long time is the following. The SA cannot distinguish a local minimum from the goal, unless it visits the very bottom of the local minimum. During the initial phase, when the temperature is high, the SA samples a large area of the configurations space, but only goes deep into local minima much later, when the temperature has been lowered. This means that, even if the robot starts at a configuration inside the goal's attraction radius, the SA will not commit to searching locally, but may jump to a distant part of the search space, and thus will visit the goal much later during the search. This is highly undesirable and it it would be advantageous to build some "opportunism" into SAPP, so that it commits itself to local explorations also in the early stages of the search.

In this section we present a modified SA algorithm, which makes use of the characteristics of the path planning problem to accelerate the search procedure.

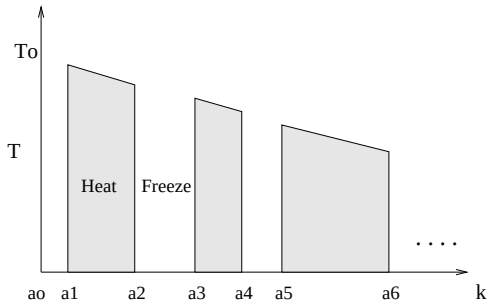

Figure 1: Freeze–heat cycles in simulated annealing.

4.1 Freeze-Heat cycles

In order to come up with a solution to the local–exploration problem, we have considered two facts. The first is that committing to a local search, means in practice to perform a gradient descent (GD). A GD is the fastest way to explore a local minimum down to its bottom. The second fact is that SA is equivalent to a GD when the temperature is close to zero. This is because the duration of the random walk is very small and the Boltzmann distribution accepts only states with lower energy. Thus, in order to add "opportunism" to the SA algorithm, it is enough to take the temperature down to nearly zero for as long as it is required to reach the bottom of the local minimum. If it is the goal, the search is finished; otherwise, the temperature can resume its original logarithmic–decay schedule. This scheme is what we call the *freeze–heat* cycle (Fig. 1). Notice that algorithmically we implement the freeze cycle as a GD. In order to prove completeness however, we treat the GD as a sequence of SA steps with zero temperature.

Freezing definitely saves time when the global minimum valley is visited. However, consider a situation where the planner has reached the bottom of a deep local minimum valley. After the temperature resumes its higher value (heating) the robot will perform a random walk. It is likely that at the end of this random walk the robot will not have escaped the local minimum attraction area. Hence, if a freeze cycle is re–applied, the planner will waste time revisiting the same local minimum. In fact, every time in the future that

the planner visits some configuration inside this minimum's attraction area, freezing will result in revisiting the same local minimum.

The problem lies in the fact that the planner cannot know, based on local information, whether it has visited the attraction area of a local minimum and thus it is not clear when it should initiate a new freeze–heat cycle. Next we propose a scheme which assists the planner in making the decision when to apply a freeze–heat cycle, i.e., points a_i, based on information collected during the search; thus, we introduce memory into the SA algorithm.

4.2 Remembering Local Minima

Ideally we would like to explore a potential minimum valley as fast as possible, escape from it as fast as possible, and never visit it again. As explained above, fast exploration is possible by freezing the temperature, or equivalently performing a gradient descent. In order to avoid revisiting the same local minima, we propose that the SA keeps in memory a list of all the local minima found so far, together with an estimate of their attraction areas.

Until now, we have used the term "attraction area" rather loosely. For our purposes, the attraction area of a local minimum $\mathbf{q}_{lm}$ is defined as the set of all configurations from which a gradient descent will terminate at $\mathbf{q}_{lm}$. Clearly such sets depend on the shape of the configurations space, and may be impossible to describe analytically. Therefore, we store in memory the rectangular *bounding box* of each set. These boxes can be computed quite easily: if gradient descent from a point $\mathbf{q}_1$ terminates at a local minimum $\mathbf{q}_{lm}$, then the boxes edges along the ith configuration axis are

$$\begin{aligned} B^i_{min} &= min\{\mathbf{q}^i_1, \mathbf{q}^i_{lm}\} \\ B^i_{max} &= max\{\mathbf{q}^i_1, \mathbf{q}^i_{lm}\} \end{aligned} \quad (7)$$

Any time later during the search that gradient descent from another point $\mathbf{q}_2$, which is outside B, terminates in a configuration inside B, the size of the bounding box B is updated (enlarged) to include $\mathbf{q}_2$, i.e.,

$$B^i_{min} = min\{B^i_{min}, \mathbf{q}^i_2\}$$

$$B^i_{max} = max\{B^i_{max}, \mathbf{q}^i_2\} \quad (8)$$

Such a scheme may overestimate the size of a local minimum attraction area. However, the ASAPP algorithm is formulated so that this does not affect the convergence properties of the SA. More specifically, in ASAPP, gradient descent will be initiated only from configurations which are outside any existing bounding box. While a configuration is within *some* bounding box, the SA will keep performing random walks. Therefore, the convergence of the ASAPP is *at least as fast* as that of the standard SAPP.

4.3 The ASAPP Algorithm

The following is the ASAPP algorithm in C–style pseudo-code.

```
k = k0;  T = T0;
q = q0;  path = q0;
done = FALSE;
while (done == FALSE) {
      if (q ∈ GOAL) {
         shortenSmooth(path);
         done = TRUE; }
      else {
           i = prevLocalMin(B, q);
           if (i == UNKNOWN) {
              q' = gradientDescent(trip,q);
              j = prevLocalMin(B, q');
              if (j == UNKNOWN) {
                 insert(B, q');
                 updateLocalMin(Bj, q');}
           else
                q' = randomWalk(trip,q);
           E = computeEnergy(q');
           ΔE = E(q') − E(q);
           if (acceptState(q', ΔE, Tk)){
              append(path, trip);
              shortenSmoothPath(path);
              q = q'; }
           k ++ ;
           Tk = T0/ ln k ;
           if (k > Kmax) {
              path = NULL;
              done = TRUE;
              } } } }
```

The ASAPP algorithm can be proved to be probabilistically complete. The proof basically shows that when performing freeze-heat cycles at points $a_o, a_1, \cdots a_i$, equation 5 (where $g_k = 0$ during freezing) still holds, i.e.,

$$\sum_{i=0}^{\infty} \sum_{k=a_{2i+1}}^{a_{2i+2}} \frac{1}{k} = \infty \qquad (9)$$

In practice, if after Kmax iterations no solution has been found the planner returns an empty path.

Next, we give a brief description of the ASAPP implementation.

5 Implementation

The robot and obstacles are represented as sets of polygons. These polygons are rasterized in a 3D bitmap which represents the workspace. All polygon intersection checks are performed by rasterizing a polygon and checking if any of its voxels was previously occupied. A robot goal configuration is defined by picking points on the robot's links and defining a set of workspace goal points for each of them. Such a scheme was proposed in [3] and these points are called *control points*. A workspace local–minima–free potential is computed for each control point by a wavefront expansion procedure. This is essentially the Manhattan distance of each control point from its goal, defined over the 3D bitmap.

The potential E for a given configuration is computed by **computeEnergy**(), which performs forward kinematics to compute the control point positions in the workspace and sums all the individual precomputed workspace potentials.

To implement the **randomWalk**(), at each state $\mathbf{q}$ a random increment of $\pm\Delta_i$, or 0 is added along q_i. This is repeated for NT steps, where T is the temperature and N a constant. Of course, at each step a collision check is performed. The **gradientDescent**() is implemented by searching randomly at each $\mathbf{q}$ for a neighbor $\mathbf{q}'$ with lower potential E. Once one is found, it becomes the next state and the descent continues. If after K trials no better state is found, $\mathbf{q}$ is treated as a local minimum. The random search for a state with lower E is necessary, because an exhaustive search requires $3^n - 1$ trials, where n is the state dimension.

The **prevLocalMin**() function checks whether $\mathbf{q}$ belongs in the bounding box B of any found local minimum; if not, **insert**() creates a new entry and implements equations 7. Function **updateLocalMin**() updates the estimate of the attraction area by using eq. 8. In **acceptState**() the Boltzmann distribution (eq. 2) is used to decide whether a trial state will be accepted. The **shortenSmoothPath**() function is applied to the current path $\{\mathbf{q}_o, \mathbf{q}_1 \cdots \mathbf{q}_k\}$ each time after a random walk, and once at the end. It starts from $\mathbf{q}_o$ and computes the maximum s for which there exists a linearly interpolated collision–free path between $\mathbf{q}_1$ and $\mathbf{q}_s$. Then ,it does the same starting at $\mathbf{q}_s$, until $\mathbf{q}_k$ is reached. The final path is a sequence of straight line segments in the state space.

The ASAPP was implemented in C and was also interfaced with a kinematic device simulation system which is used for workcell design and off-line programming of industrial robots. The functionalities offered by this system are accesible by a graphical user interface, or by a command language interpreter. Once a workcell description has been loaded, the ASAPP can be invoked from a pull–down menu. The user can use the mouse to select the robot(s) for which planning is to be performed, and the obstacles. By default, each robot's single control point is at its tool–tip. However, any number of control points, on any link, can be defined. The goal set for each control point is a set of polygons that is selected by the user. The ASAPP is launched as a separate process because its running time can be in the order of tens of minutes.

6 Simulation Results

We have constructed two planar path planning examples of a triangular robot amidst polygonal obstacles (Fig. 2). The control point was the right lower corner of the triangle–robot and the goal was the bottom left corner of the workspace. Path planning was performed 1000 times to collect some statistics. The size of the bitmap was

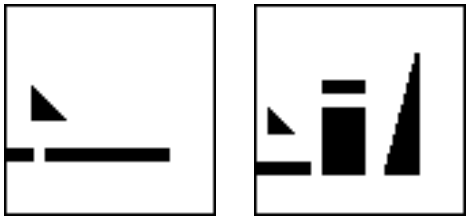

Figure 2: 2D examples with 1 and 3 local minima.

128x128 pixels. There were 1 and 3 local minima in the left (first) and second (right) examples respectively. In the first example both the SAPP and ASAPP algortihms were used, in order to compare their performance. The SAPP required an average of $T_{av} = 111sec$ with minimum and maximum times being 10 and 2104 sec respectively. For the ASAPP these times were 2, 14 and 76 sec, respectively. Clearly, the ASAPP is about an order of magnitude faster . The ratio T_{max}/T_{av} is also 5.4, while for SAPP it is 19. This means that the ASAPP running times are more consistent, with less variance. The second example was ran only for ASAPP, with $T_{min} = 1sec$, $T_{av} = 35sec$ and $T_{max} = 243sec$. The average time is higher because there are 3, and not 1 local minima to be explored.

A more interesting experiment was to plan the motion for 3 planar robots (9 DOF's) in a narrow corridor (see Fig. 3). The initial state is shown in the top–left frame and the goal state in the bottom–right frame (time increases from left to right). The bitmap used was 256x256. For each robot, the control point was a polygon vertex and hence the orientation at the goal is arbitrary. This is a difficult problem with the initial state being at a very deep local minimum because the potentials of the irregular and rectangular robots drive them to collide with the triangle robot; however, both of them eventually back–up in order to interchange positions. The planner was ran for 1000 times and the running times were $T_{min} = 29sec$, $T_{av} = 94.3sec$ and $T_{max} = 354sec$. The T_{max}/T_{av} ratio was 3.75. All examples were ran on a SUN SPARC10 machine and timed with the C language **time**() function. Notice that in this example there exist also other paths towards the goal (e.g. a different back–up sequence) and

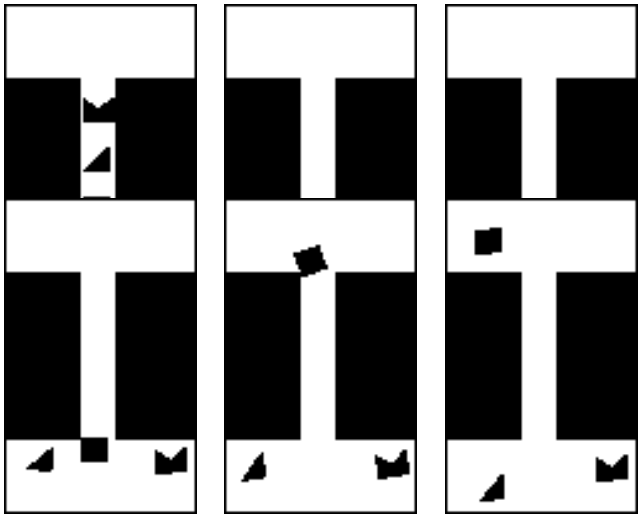

Figure 3: Coordinated motion of three 3–DOF robots.

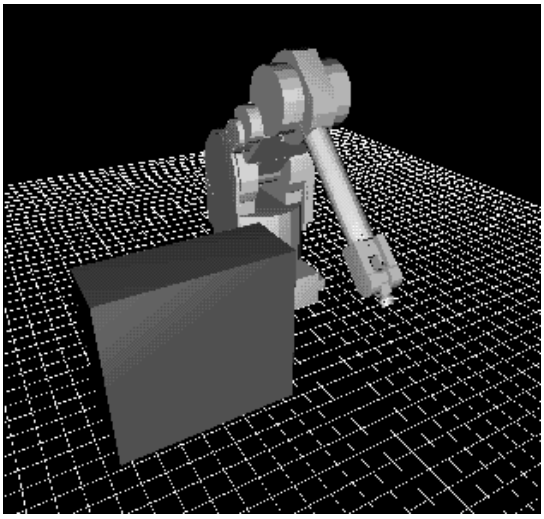

Figure 4: COMAU robot example.

thus, the reported times are computed over possibly different state-space paths.

We have also interfaced the planning algorithm with a 3D robot simulator. We are currently conducting preliminary experiments involving a 6 DOF COMAU industrial robot, with a geometric model consisting of 630 polygons. The workspace is discretized in a 3D array of size 128 x 128 x 128. In one of these experiments, the robot was commanded to move its tool–tip to a specific 3D location, behind a box–obstacle, while checking for self–intersection among all its links (and for collision with the obstacle).

In 20 simulations the running times were computed to be $T_{min} = 61sec$, $T_{av} = 291.4sec$ and

$T_{max} = 638sec$. The computation of the potential field over this array required 54 *sec* and is included in the numbers for the running times. The goal of this simulation was not to produce a complex path, but rather to get a feeling for the running times for more complex robot models. The preliminary results are encouraging and suggest that path planning for more complex environments may be feasible using ASAPP.

7 Discussion and Future Work

We have presented a path planning methodology which uses a novel accelerated simulated annealing optimization algorithm as a search mechanism for finding a path towards the global minimum of an artificial potential field. The algorithm was shown to be capable of escaping from deep local minima and of generating paths for high–dimensional problems (9 DOF's) involving multiple robots. The planner has been interfaced with a 3D robot simulation package. We are currently evaluating the ASAPP performance in real–world problems involving robot and obstacle representations with hundreds of polygons.

One drawback of the proposed planner is that it is only probabilistically complete and its running time is a random variable. We plan to estimate the running time distributions for different classes of problems and determine how long the planner should be allowed to run, before declaring with adequate confidence that no solution exists. Also, parallel implementations of the ASAPP could be used to decrease the T_{av}/T_{min} ratio.

Another practical constraint of the current ASAPP implementation is that the robot workspace needs to be digitized, in order to compute the individual control–point potentials and to perform intersections. We have ran examples with 3D arrays of size 128x128x128. However, 3D arrays become impractical when high resolution is needed over large workspaces. Work is underway to use an octree representation instead of a 3D array, thus sacrificing speed for spatial resolution. Finally, the path found by ASAPP–and any random path planner–will be different each time the planner is executed. If this is undesirable it may possible to avoid regions of the state space by imposing extra constraints.

References

[1] Jérôme Barraquand. A monte–carlo algorithm for path planning with many degrees of freedom. In *Proc. of the IEEE Intern'l Conf. on R & A*, pages 1712–1717, 1990.

[2] Jérôme Barraquand. Robot motion planning: A distributed representation approach. *Int. J. of Robotics Res.*, 10(6):628–649, 1991.

[3] Jérôme Barraquand, Bruno Langlois, and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. Technical Report STAN-CS-89-1257, Dept. of Computer Science, Stanford University, 1989.

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

[5] C.I. Connoly, J.B. Burns, and R. Weiss. Path planning using Laplace's equation. In *Proc. of the IEEE Intern'l Conf. on R & A*, pages 2102–2106, 1990.

[6] B. Donald. Motion planning with six degrees of freedom. Technical Report AI-TR-791, MIT, 1984.

[7] Lester Ingber. Simulated annealing: Practice versus theory. *Journal of Mathematical and Computer Modelling*, 18(11):29–57, December 1993.

[8] D. E. Koditschek and E. Rimon. Exact robot navigation using cost functions : The case of distinct spherical boundaries in e^n. Technical report, Yale, 1988.

[9] A. Papoulis. *Probability, random variables, and stochastic processes*. McGraw-Hill, 1984.

[10] C. Gelatt Jr. S. Kirkpatrick and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[11] J. T. Schwarz and M. Sharir. On the 'piano movers' problem: I. the case of a two–dimensional rigid polygonal body moving amidst polygonal barriers. *Comm. on Pure and Applied Mathematics*, 34:345–398, 1983.

[12] J. T. Schwarz and M. Sharir. *On the 'Piano Movers' Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds*, pages 298–351. Academic Press 4, 1983.

[13] H. Szu and R. Hartley. Fast simulated annealing. *Physical Letters*, 122(3-4):157–162, 1987.