Towards Real-Time Execution of Motion Tasks

Sean Quinlan and Oussama Khatib

Robotics Laboratory
Computer Science Department
Stanford University

Abstract

The goal of the research described in this paper is to build robotic systems that can execute motion tasks in real time. We present two ideas towards this goal: a real-time path planner for static three-dimensional configuration spaces and a new framework for the integration of planning and control. The planner is based on a new class of cells, *slippery cells*, incorporated in an approximate cell decomposition algorithm. The *elastic band* concept is proposed to provide an effective link between planning and execution. With elastic bands, a path is treated as a flexible entity. The initial configuration of the elastic band is the path provided by the planner. The shape of the elastic dynamically evolves during execution, using sensory data about the environment.

1 Introduction

The goal of the research described in this paper is to build robotic systems that can execute motion tasks in real time. The desired system would fulfill commands to move to a desired position while avoiding obstacles in the environment. Our additional goal of achieving real-time performance relates to the responsiveness of the system; the time to execute the task is of importance and is both a motivating force in the search for new ideas and a means of comparing alternative solutions.

To execute a motion task a robot must combine the ability to plan motions and to execute them. In addition, the robot has sensors that enable it to monitor the effect of its actions and to perceive the environment. One approach to building robot systems is illustrated in Figure 1. In this approach, the robot uses its sensors to build a world model that is passed to a path planner. A task from the user is specified to the path planner which generates a trajectory for the robot to follow. A motion controller is used to track the path using feedback from sensory information of position and velocity. Three observations can be made about the above scheme:

First, with this approach, the responsiveness of the robot is totally dependent on the speed of the path planner. Complete solutions to path planning are computational complex. In recent years, much progress has been made in developing practical algorithms which

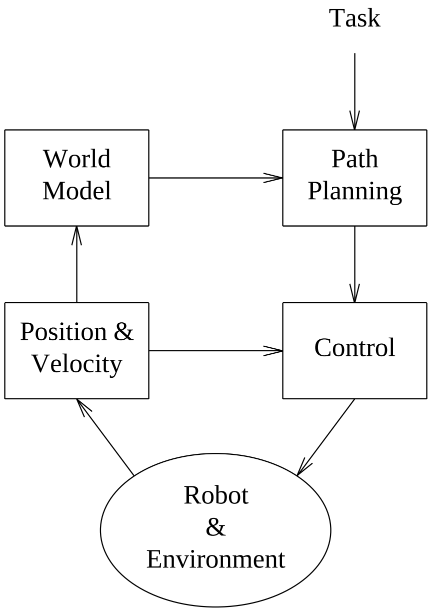

Figure 1: One scheme for a robot system

are efficient (Latombe 1991, Barraquand and Latombe 1991, Lengyel et. al. 1990). For typical three-dimensional problems, current implementations can generate a path in several seconds.

Second, it is a non-trivial problem to connect a path planner and a control system. A common approach is the use of trajectories generated by time parameterizing the path produced by the planner. A dynamic model of the robot can be used to insure that the generated trajectory is feasible or, better yet, optimal (Bobrow, Dubowsky, and Gibson 1983). Unfortunately, path planners are often designed to find any feasible path, with little attention to its suitability for execution. Planners may try to produce kinematically short paths but rarely do they use any knowledge of the robot dynamics. The result is that the path may be undesirable from a control point of view. For example, the path may have abrupt changes in direction or maintain little clearance from obstacles, thus requiring the robot to move slowly. Optimally short paths suffer substantially from this problem.

Third, there is an underlying assumption that the motion plan will be valid for the time it takes the robot to execute it. The validity of this assumption depends on the degree to which the world model represents the environment. At one extreme, there is a complete model of a static environment and thus a path can be used blindly. At the other extreme, there is no model of the changing environment and thus the concept of a preplanned path is meaningless. In reality, the situation is somewhere in between: there is some a priori knowledge that the planner can use and, during execution, the robot uses sensors to detect unexpected obstacles and uncertainties in the environment.

In the scheme shown in Figure 1, sensory information is used to build a world model; if changes in the environment are detected during execution then the world model will change. Under this scheme, if the world model is changed during execution, the obvious action is to halt the robot and plan a new path. In such situations, the iteration through the world modeler and path planner can be considered as a feedback loop. This loop, even with fast path planners, is expected to be the bottleneck in building responsive systems.

The first part of this paper describes a real-time path planner for static three-dimensional configuration spaces. This planner is based on a variation of the approximate cell decomposition method. The method uses a new class of cells, called *slippery cells*, that have a general shape and can typically describe the free space with fewer cells than other approximate cell decompositions. The result is a corresponding reduction in the required search time.

The second part of this paper describes a new framework to deal with the second and third observations above. In this framework, an intermediate level based on the *elastic band* concept is created between the planner and controller. With the elastic band, a path is treated as a flexible entity. Modifications to the path are made incrementally, using an artificial potential field (Khatib 1986). These modifications use knowledge about the obstacles in the environment, the dynamics of the robot, and sensory data.

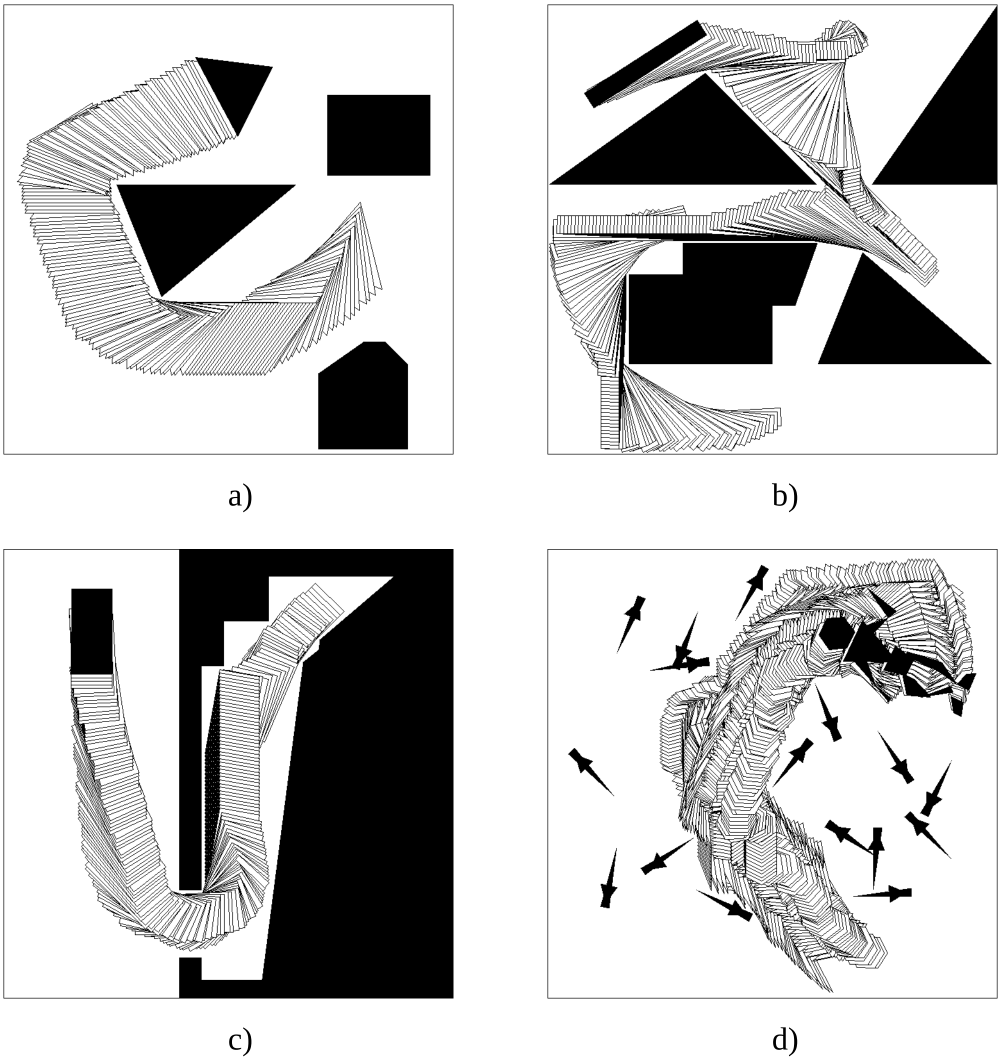

Figure 2: Some examples of paths generated by the planner

2 A Real-Time Planner

This section describes a path planner that operates in real time for static three-dimensional configuration spaces. Problems such as those shown in Figure 2 were solved by the planner in about one tenth of a second on a DECstation 5000.

The planner is based on the approximate cell decomposition method (Brooks and Lozano-Pérez 1985, Zhu and Latombe 1991). The robot's free space is decomposed into simple regions, called *cells*, such that a path between any two configurations in a cell can easily be generated. A connectivity graph of the cells is constructed and searched; the result is a sequence of cells from which a path can be computed. The planner is approximate in that the union of the cells may be a subset of the free space. The free space is represented to a resolution; the higher the resolution, the more accurate the approximation becomes. The effect of this approximation is that the planner may fail to find a path when one exists.

Slippery Cells

The novel feature of this planner is the use of a new class of cells, called *slippery cells*. The shape of a slippery cell is general, yet it maintains the property that a path between any two configurations in the cell can be generated quickly using a local method. Using slippery cells, the free space can typically be described with fewer cells than implementations that use rectangular cells. The connectivity graph is smaller with the result that the search time is reduced.

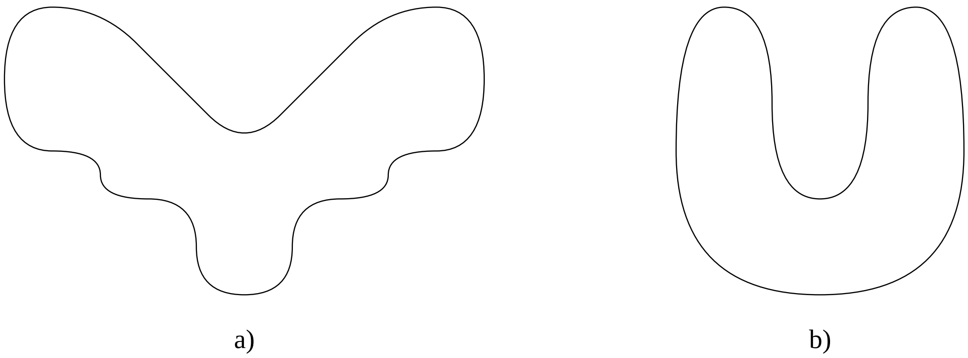

Figure 3: a) A slippery cell. b) A cell that is not slippery

Definition A cell is slippery if the outward normal from any point on the boundary does not intersect the cell.

Figure 3 shows two cells; the first is slippery, the second is not. The important property of slippery cells is that any two points in the cell can be connected by a path using an algorithm that is local. Although the proof is not given here, the intuition is as follows. To get from point A to point B, move in a straight line towards B until either B or the cell boundary is reached. In the case where the boundary is reached first, slide along the boundary by projecting the desired motion towards B into the plane tangential to the boundary. If the motion vector towards B points away from the boundary then stop sliding and move directly towards B. As the outward normal does not intersect the cell, the vector of desired motion towards B will always contain a component tangential to the boundary. It impossible to become "stuck" while sliding on the surface; the cell is slippery. It can also be shown that the motion monotonically decreases the distance to B and thus will reach its goal. Figure 4 gives an example of a path within a slippery cell.

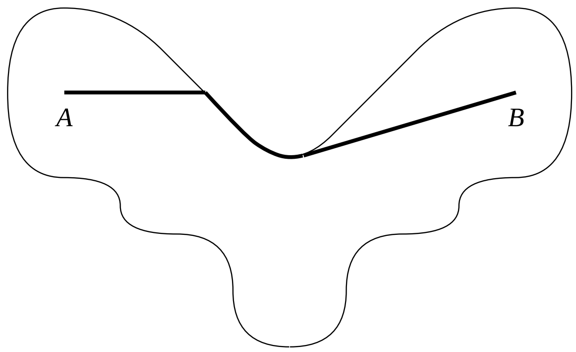

Figure 4: A path in a slippery cell

Free Space Decomposition

To use slippery cells in a planner, an algorithm is needed to decompose the free space into a set of slippery cells. As with other approximate cell decompositions, the set of cells should be disjoint and represent the free space as closely as possible. Instead of approaching this problem directly, we use an intermediate representation based on bitmaps. The bitmap configuration space, which is approximate, is then decomposed into slippery cells.

A bitmap representation is constructed by dividing the configuration space into a grid of equal sized rectangular cells. Each cells is labeled *free* or *forbidden*. Free cells are contained entirely in the free space, while forbidden cells intersect a configuration space obstacle. Such a representation can be stored in a computer as an array of bits of the appropriate dimension, hence the name bitmap. A bitmap representation is an approximate cell decomposition of the configuration space and is accurate to the resolution of the grid. For example, the three dimensional configuration spaces used to solve the problems in Figure 2 were represented as a $128 \times 128 \times 128$ grid of cells using a $128 \times 128 \times 128$ array of bits.

This paper does not address the issue of how to compute a bitmap configuration space for a robot in an environment, but several approaches have been proposed (Branicky and Newman 1990; Lengyel et. al. 1990). One scheme constructs the bitmap from an analytical description of the configuration space by rendering the obstacles in an operation similar to drawing a graphical image. For the remainder of this paper we will assume that a bitmap configuration space has been generated for the environment.

To describe the bitmap configuration space as a set of disjoint slippery cells, each free bitmap cell is labeled with an integer indicating the slippery cell to which it belongs. The idea behind the labeling algorithm is the following. Find a free bitmap cell that has not been labeled as part of a slippery cell. This cell becomes the seed of a new slippery cell. The seed is grown by adding neighboring unlabeled bitmap cells that are in the free space, while preserving the slippery cell property. When no more bitmap cells can be added, the process is repeated to generate another slippery cell. If no unlabeled free bitmap cells are found, then the algorithm is complete.

The abstruse aspect of the algorithm is determining whether an unlabeled free bitmap cell may be added to a slippery cell. A simple, but computationally expensive solution is to check all bitmap cells that intersect outward normals of the bitmap cell in question. Surprisingly, we have found a procedure that determines this condition in a single memory access. Although the procedure is simple, the proof of correctness is involved and is omitted here.

The following procedure will decide if a bitmap cell can be added to a slippery cell to produce a new slippery cell. Determine the faces of the bitmap cell which will be connected to the slippery cell; the number of these faces can vary from one to the dimension of the configuration space. The set of normals of these faces define a subspace. Project the slippery cell and the bitmap cell into the subspace. If the projections are disjoint then the bitmap cell may be added to the slippery cell.

To implement the above procedure efficiently, the projections of the slippery cell into all possible subspaces are maintained while the slippery cell is grown. When a bitmap cell is added to the slippery cell, the projections are also updated. To determine whether a particular bitmap cell may be added, a single memory access into the appropriate projection is all that is needed.

To give a more concrete description of the entire algorithm, the pseudo code for the two dimensional case is given below. The input to the algorithm is a two dimensional array, C, describing the configuration space as a set of free and forbidden bitmap cells. The output is the same array, with each free bitmap cell labeled by the slippery cell to which it belongs. The following convention is used for the elements of C: negative one is a forbidden cell, zero is a free cell, and $i > 0$ is part of the ith slippery cell. Two one dimensional arrays, X and Y, record the projection of the current slippery cell onto the x-axis and y-axis respectively. A list, l, queues the bitmap cells to be examined.

a) b)

Figure 5: Two decomposition of a two dimensional configuration space into slippery cells. The dark region represent the obstacles and the lines denote the boundaries of the cells.

```
procedure slippery;
begin
    cell ← 0;
    for each element q in C do begin
        if C[q] = 0 then begin
            cell ← cell + 1;
            initialize l to q;
            while l is not empty do begin
                p ← first element of l;
                if check(p) then begin
                    C[p] ← cell;
                    X[p_x] ← cell;
                    Y[p_y] ← cell;
                    append the four neighbors of p to l;
                end
            end
        end
    end
end;

function check(q) : Boolean;
begin
    /*
     * Determine the axes along which the bitmap cell will connect
     * to the slippery cell
     */
    xaxis ← C[q+(1,0)] = cell or C[q+(−1,0)] = cell;
    yaxis ← C[q+(0,1)] = cell or C[q+(0,−1)] = cell;

    /* Check the appropriate projection */
    if xaxis and yaxis then
        return C[q] ≠ cell;
    if xaxis then
        return X[q_x] ≠ cell;
    if yaxis then
        return Y[q_y] ≠ cell;

    /* The bitmap cell must be a seed and is thus a slippery cell */
    return true;
end;
```

The decomposition into slippery cells produced by the above algorithm is neither unique nor optimal. Figure 5 gives an example of two decompositions produced for a two dimensional configuration space. The different decompositions are the result of scanning the cells of the bitmap configuration space in a different order and thus different bitmap cells are selected as the seeds for slippery cells. Although a given decomposition is not optimal, it can be shown that each slippery cell is maximal under the constraints of the configuration space obstacles and the previously grown slippery cells; for a given slippery cell, there is no larger slippery cell that completely covers the other cell but does not intersect the bitmap cell that are forbidden or already labeled as part of another slippery cell.

The algorithm is efficient and generates a small number of cells. The time to examine a single bitmap cell is $O(1)$ and each cell is examined at most $2d$ times, where d is

Table 1: Number of cells for the examples in Figure 2.

Example	*Free Bitmap Cells*	*Slippery Cells*	*Octree Method*
a	1238839	500	50179
b	628991	134	22781
c	154521	408	45405
d	114097	1010	31171

the dimension of the configuration space. For a bitmap with n cells, the total time to generate the slippery cells is $O(dn)$. The examples shown in Figure 2 were generated with a $128 \times 128 \times 128$ bitmap and the computation of the slippery cells took about twenty seconds on a DECstation 5000. For these examples, Table 1 compares the number of slippery cells with the much larger number of cells produced by an Octree decomposition at the same resolution (Faverjon 1984).

The Path Planner

After generating a set of slippery cells, a connectivity graph is constructed. The nodes of the graph represent the slippery cells while the arcs represent the adjacent cells. The graph can be constructed by scanning the array in which the bitmap cells are label. The scanning is an operation of order $O(dn)$ and, for the examples in Figure 2, takes about five seconds on a DECstation 5000. Table 2 gives the number of arcs in the connectivity graphs and the average number of connections per cell. As can be seen, the number of connections is low and appears to be a result of the typical size distribution of slippery cells. There tends to be a small number of large cells that describe the vast majority of the free space. Around these large cells lie many tiny cells that describe the complex surfaces of the configuration space obstacles. These tiny cells are often only connected to a single cell, namely their neighboring large cell.

Table 2: Number of in the connectivity graphs

Example	*Number of Arcs*	*Average connectivity*
a	1059	2.12
b	211	1.57
c	651	1.60
d	1673	1.66

A path between a start and goal configuration is computed in three steps. First, a sequence of adjacent slippery cells is found such that the start and goal configurations are enclosed in the first and last cell respectively. Second, for each pair of adjacent cells in the sequence, a via point is found that lies on the boundary of both cells. Third, a path is computed by concatenating segments of path between consecutive via points. These steps are the same as most other cell decomposition methods.

A sequence of cells is created by searching the connectivity graph. Ideally, the sequence should contain the optimal path and, in the context of path planning, the optimal path is typically the shortest path. However, finding such a sequence is a difficult problem; instead, in this planner, we use a heuristic intended to generate a "good" sequence. In particular, we generate the sequence that has the least number of cells by using a breadth first search. Many other heuristics are possible, for example minimizing the volume of the cells in the sequence, and we intend to investigate their relative merits.

The time to search a graph is order $O(e)$ where e is the number of arcs in the graph. For the examples in Figure 2, the connectivity graphs are small and the search time is less than one hundredth of a second.

The selection of via points also influences the length of the resulting path. The simplest solution is to have a single, predetermined via point for each pair of adjacent slippery cells. However, as some slippery cells are large, the fixed via points could lead to long, unnecessary detours. The current implementation determines the via points using a greedy search. From the start configuration, the closest point is found on the boundary between the current cell and the next cell in the sequence. This point becomes the first via point. The second via point is found in the same manner, except that the first via point is now the position from which the distance is calculated. This is repeated until the last cell is reached. As with any greedy search, the result is not optimal but produces reasonable results. The implementation of the greedy search requires many candidate boundary points to be examined. For the examples in Figure 2, the selection of via points takes about one tenth of a second.

Finally, the via points are connected to form a path. Path segments are found between the start configuration and the first via point, the final via point the goal, and each pair of consecutive via points. Each path segment lies entirely within a slippery cell and can be generated using the algorithm, described previously, for finding such paths. The path segments are then concatenated to form the complete path. The time to generate the path segments is proportional to the length of the path. For the examples in Figure 2, the path was constructed in about one hundredth of a second.

For a static, three-dimensional configuration space that is represented to a reasonable resolution, the planner described here can be considered to operate in real time. If the environment in which a robot moves does not change, then the configuration space will be static. The formation of a set of slippery cells and the generation of a connectivity map are then precomputation steps. When a task is specified, a path can be generated by searching the connectivity graph, determining via points, and concatenating path segments. Additional paths can be generated by repeating these steps without recomputing a decomposition of the configuration space. For a three dimensional configuration space represented to a resolution of $128 \times 128 \times 128$, the total time to generate a path is a fraction of a second.

The major limitation of this planner is the large memory it requires. For a three-dimensional configuration space, represented to a resolution of $128 \times 128 \times 128$, about four megabytes of memory are needed. At the same resolution, a six-dimensional configuration space requires sixteen terabytes of memory. The memory requirements restrict the planner to three or four dimensional configuration spaces.

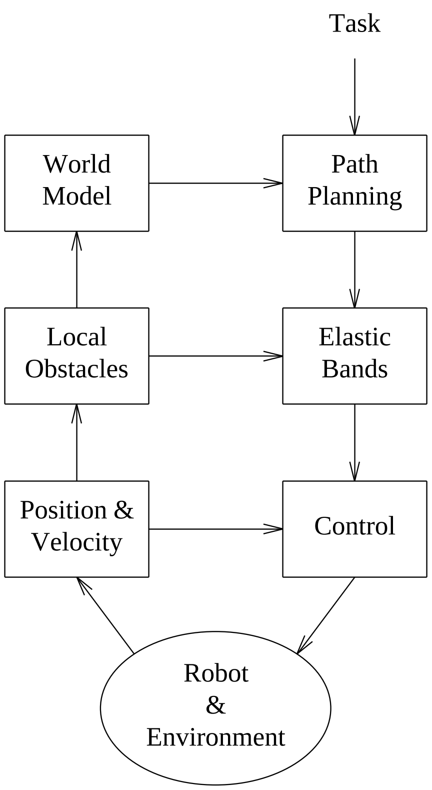

Figure 6: Proposed structure for a robotics system

3 Elastic Bands

The elastic band concept is aimed at providing an effective link between planning and execution. With the elastic band, a path is treated as a flexible entity. The initial configuration of the elastic band is the path provided by the planner. The shape of the elastic dynamically evolves during execution, using sensory data about the environment. This provide a tight connection between the robot and its environment while preserving the global knowledge needed to solve general tasks.

In this framework, the robot system has the structure shown in Figure 6. The system is a three-level hierarchy, with each level forming a closed loop with the environment. Global path planning at the highest level with the longest time cycle, the elastic band in the middle to handle local sensory data about the environment, and a controller at the bottom that closes the loop with the robots actuators and associated sensors.

The elastic band is modified by subjecting it to an artificial field of forces. These forces are designed to affect the path's length, smoothness, and obstacle clearance. The changes in the shape of the elastic band are incremental. These changes are restricted to maintain the elastic in the free space and thus it remains a globally free path.

For example, the application of an internal contraction force could be used to remove any slack and reduce the length of the elastic band. Since the elastic band is to remain in the free space, the elastic will shorten until it lies on the boundary of the configuration space obstacles. Figure 7-a shows a path obtained from a planner in a two-dimensional configuration space. This path is used as the initial configuration for an elastic band. Figure 7-b shows the elastic band after the application of an internal contraction force.

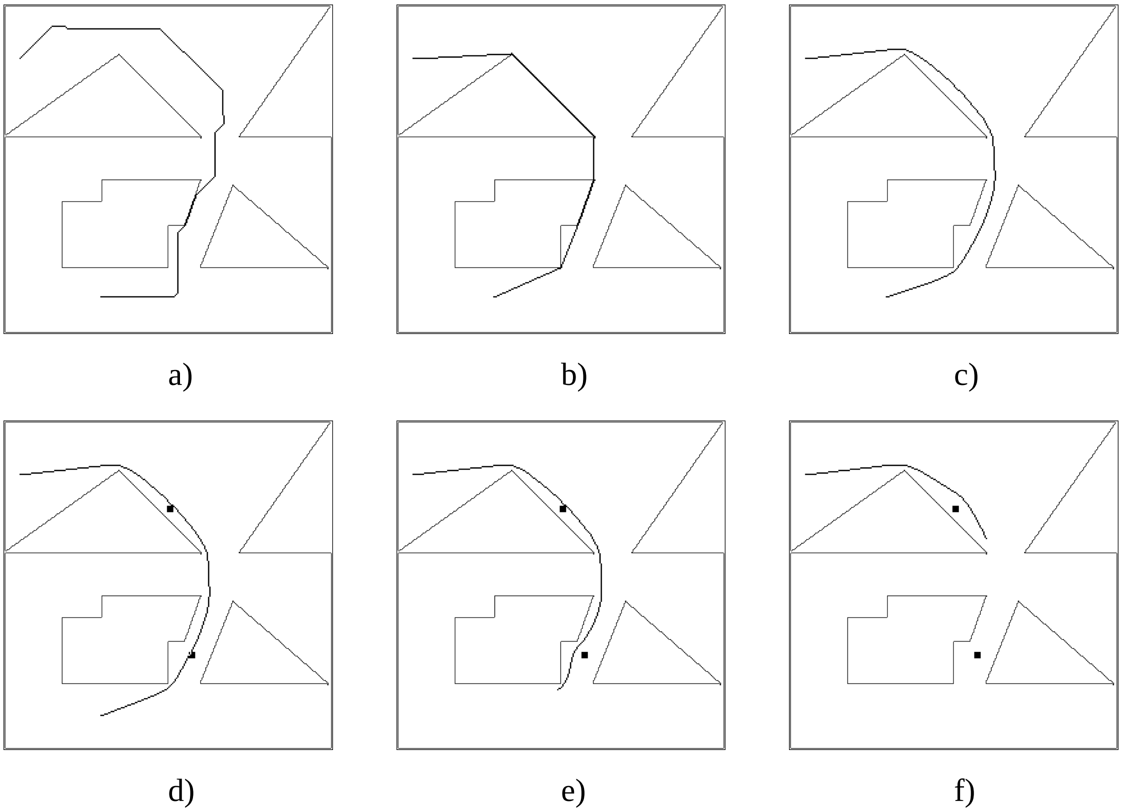

Figure 7: a) A path generated by a planner. b) Applying an internal contraction force. c) Applying both an internal contraction force and an external repulsion force. d) Some unknown obstacles on the path. e) and f) Avoiding the unknown obstacles during execution.

The internal contraction forces produce paths that are locally optimal in length. However, these paths lie on the boundaries of the configuration space obstacles and may exhibit discontinuities in direction. Obstacle clearance and first order continuity of the elastic are required for the trajectory tracking level. One method to achieve both properties is to apply a repulsive field of forces. Each point on the elastic band is repelled from the closest point on the configuration space obstacles' boundaries. For the above example, the application of such a force field is shown in Figure 7-c.

As the robot moves along the elastic band, sensory information is used to deform the shape of the elastic while maintaining a path in the free space to the goal. This mechanism allows the robot to deal with unexpected obstacles. When such obstacles are detected, repulsive forces are applied to deform the elastic and to provide collision avoidance. Figure 7-d shows two unknown obstacles along the elastic. Figures 7-e and 7-f show the result of moving along such an elastic. As the robot approached the obstacles, the sensors detect them and the elastic is deformed.

Obviously, if the changes in the environment are large, the elastic band could fail to produce a feasible path even if one exists. This problem is typical of local collision avoidance methods and is the primary reason that path planning is needed. In such a situation, the failure can be detected and a new path can be found by replanning.

Implementation

The continuous curve of an elastic band is approximated by set of points in the configuration space. Forces are applied to these points and the effect is simulated by numerical integration. The elastic band, at any given instant, is constructed by concatenating paths between the points.

Each point, $\mathbf{p}_i$, is subjected to the following three forces.

- A contraction force that models the effect of connecting a spring between each point on the elastic band. The force is determined by the equation

$$\mathbf{f}_c = -k_c\,(\mathbf{p}_i - \frac{\mathbf{p}_{i+1} + \mathbf{p}_{i-1}}{2}).$$

  where k_c is the contraction gain.

- A repulsive force from the obstacles in the environment given by the equation

$$\mathbf{f}_r = \begin{cases} k_r(\frac{1}{\rho} - \frac{1}{\rho_0})\frac{1}{\rho^2}\frac{\partial \rho}{\partial \mathbf{p}} & \text{if } \rho \leq \rho_0 \\ 0 & \text{if } \rho > \rho_0 \end{cases}$$

  where k_r is the repulsive gain, $\rho(\mathbf{p})$ is the distance at the point $\mathbf{p}$ to the nearest obstacle, and ρ_0 is the maximum distance at which the repulsive is applied. $\frac{\partial \rho}{\partial \mathbf{p}}$ denotes the partial derivative vector of the distance from $\mathbf{p}$ to the obstacle

$$\frac{\partial \rho}{\partial \mathbf{p}} = [\frac{\partial \rho}{\partial x}\ \frac{\partial \rho}{\partial y}\ \frac{\partial \rho}{\partial z}]^T.$$

  If $\mathbf{q}$ is the position of the closest obstacle, the distance ρ is given by $\rho = \mid \mathbf{p} - \mathbf{q} \mid$ and

$$\frac{\partial \rho}{\partial \mathbf{p}} = \frac{\mathbf{p} - \mathbf{q}}{\rho}$$

- A velocity damping force given by

$$\mathbf{f}_v = -k_v \dot{\mathbf{p}}_i$$

  where k_v is a velocity damping gain. The damping force decreases the settling time for the elastic band.

To ensure that the elastic band remains in the free space, a bitmap representation of the configuration space is used[1]. The points are restricted to the free bitmap cells and each

[1] See the previous section for details of bitmap configuration spaces

point must maintain a distance from the two neighboring points that is less than the width of a bitmap cell. These restrictions enable a continuous curve that is enclosed in the free space to be constructed between neighboring points.

For the static portion of the environment, the distance ρ and its derivatives are efficiently computed using a lookup table. Sensory data is used to deal with the dynamic part of the environment and unexpected obstacles. In the current implementation, the shapes of unexpected obstacles are restricted to hyper-spheres.

The total time to compute the force for a point is $O(u)$, where u is the number of unexpected obstacles. For a small number of unexpected obstacles, our current implementation can perform about one hundred thousand point evaluations a second. A typical path can be described with less than 1000 points, giving a servo-rate of about one hundred hertz.

The major limitation of the current implementation is the memory required for a bitmap representation of the configuration space. With current technology, such a representation is feasible only for configuration spaces of dimensions of three or four. We are investigating techniques that do not require an explicit representation of the entire configuration space.

4 Conclusion

Planning is expected to be the bottleneck in building systems that can execute motion tasks in real time. This paper describes a real-time path planner for static three-dimensional configuration spaces. The planner achieves this speed by precomputing a compact description of the configuration space using slippery cells. The shape of a slippery cell is general, yet it maintains the property that a path between any two configurations in the cell can be generated using a local method. With slippery cells, the free space is generally described with fewer cells than implementations that use rectangular cells. The connectivity graph is smaller with the result that the search time is reduced. After decomposition of the free space, the planner can compute paths between start and goal configurations in a fraction of a second on a typical workstation.

Elastic bands constitutes an effective framework for dealing with real-time collision-free motion control for a robot operating in an evolving environment. A planner provides an initial path that is a solution to the problem of moving a robot between a start and goal configuration. Incremental adjustments to the path are made while maintaining a global path in the free space. These modifications are based on sensory data about the environment and desired criteria concerning the path, such as length, smoothness, and obstacle clearance. Implemented as a real-time servo-loop, an elastic band provides many of the benefits of reactive systems without sacrificing global planning.

5 Acknowledgments

This material is based upon work supported under a National Science Foundation Graduate Fellowship and DARPA contract (DAAA21-89-C0002). Many thanks to Jim Bobrow, Jean-Claude Latombe, Ross Quinlan, and Mark Yim for comments and suggestions on the work described here.

6 References

Barraquand, J. and Latombe, J. C. 1991. "Robot Motion Planning: A Distributed Representation Approach," *The International Journal of Robotics Research,* MIT Press, 10(6).

Bobrow, J., Dubowsky, S., and Gibson, J. 1983. "On the Optimal Control of Robotic Manipulator with Actuator Constraints," *Proc. American Control Conference,* San Francisco, pp. 782-787.

Branicky, M. S. and Newman, W. S. 1990. "Rapid Computation of Configuration Space Obstacles," *Proc. IEEE International Conference on Robotics and Automation,* Cincinatti.

Brooks, R. A., and Lozano-Pérez, T. 1985. "A Subdivision Algorithm in Configuration Space for Find-Path with Rotation," *IEEE Transactions on Systems, Man, and Cybernetics,* SMC-15 (2), pp. 244-233

Faverjon, A. B. 1984. "Obstacle Avoidance using an Octree in the Configuration Space of a Manipulator," *Proc. IEEE International Conference on Robotics and Automation,* Atlanta. pp. 504-512

Khatib, O. "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotic Research,* vol. 5, no. 1, Spring 1986, pp. 90-98.

Latombe, J. C. 1991. *Robot Motion Planning,* Kluwer Academic Publishers, Boston.

Lengyel, J. Reichert, M., Donald, B. R., and Greenberg, D. P. 1990. "Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware," *Proc. SIG-Graph,* Dallas.

Zhu, D. and Latombe, J. C. 1991. "New Heuristic Algorithms for Efficient Hierarchical Path Planning," *IEEE Transactions on Robotics and Automation,* Vol. 7, No. 1. pp. 9-20.