IEEE Transactions on Robotics and Automation, 11(3), 1995, 408-413

Computation of Configuration-Space Obstacles Using the Fast Fourier Transform

Lydia Kavraki
kavraki@cs.stanford.edu
Robotics Laboratory, Department of Computer Science,
Stanford University, Stanford, CA 94305, USA

Abstract

This paper presents a new method for computing the configuration-space map of obstacles that is used in motion-planning algorithms. The method derives from the observation that, when the robot is a rigid object that can only translate, the configuration space is a convolution of the workspace and the robot. This convolution is computed with the use of the Fast Fourier Transform (FFT) algorithm. The method is particularly promising for workspaces with many and/or complicated obstacles, or when the shape of the robot is not simple. It is an inherently parallel method that can significantly benefit from existing experience and hardware on the FFT.

Keywords: Configuration Space, Fast Fourier Transform, Collision Checking, Robot Motion Planning

1. Introduction

The problem of planning the motion of a robot among static obstacles has been recently approached by restricting the configuration space (C-space) of the robot to a discrete space, where each degree of freedom can only assume a finite number of values [2]. In this approach, it can be very useful to precompute and store a C-space bitmap that explicitly represents the free part of the C-space (the 0's) versus the part that gives rise to collision with an obstacle (the 1's). This bitmap reduces collision checking to an $O(1)$ operation [11]. In fact, there is a variety of other ways a precomputed C-space bitmap can be exploited by a planner. For example, in [14] it is used by a wavefront propagation algorithm to numerically compute a local-minima-free potential field with a single minimum at the goal. Another possible use of the bitmap is to generate a more concise representation of the free space by grouping adjacent free configurations into hyperparallelepipeds of various sizes ('approximate cell decomposition' approach to path planning [4, 27]).

We describe below a new method for computing the C-space bitmap. In its most basic form, it applies to the case where the robot is an n-dimensional ($n = 2$ or $n = 3$) rigid object translating in an n-dimensional workspace among obstacles. The C-space, which in this case is also n-dimensional, can be regarded as a convolution of the workspace and the robot [6]. We compute this convolution via a Fast Fourier Transform (FFT) algorithm. The running time of our algorithm depends only on the resolution of the discretization used. For a fixed resolution it is independent of the complexity and the shape of the robot and the obstacles. Moreover, the method can benefit directly from specific hardware developed for the FFT algorithm.

The same method can be extended to a robot that can both translate and rotate in the plane by discretizing the range of orientations of the robot and building a three-dimensional C-space bitmap slice by slice. In theory, this extension also applies to a robot that translates and rotates in three dimensions, but then the dimension (6) of the C-space makes the construction of an explicit bitmap unrealistic. However, if the rotation is restricted to occur about a single axis, a relatively coarse four-dimensional bitmap can be constructed as a set of three-dimensional slices, each computed at a fixed orientation of the robot. If the robot is a planar set of K rigid bodies connected by prismatic and revolute joints, the method can be used to build K bitmaps, each representing the C-space of one of the bodies as if it were free to translate and rotate. In the context of the planning approach described in [2], these K bitmaps can be used to compute whether a configuration of the robot is collision-free or not in $O(K)$ time.

Although there has already been considerable research and results in C-space computation, fast computation of C-space obstacles remains an important issue. This is in particular the case when the environment changes dynamically. We believe that C-space computation is basic enough in robotics (and other domains) to make it benefit from parallel or specific hardware implementations. In that respect, bitmap representations seem to be very suitable. An analogy can be drawn with some basic graphics problems (e.g., hidden surface elimination), which admit a relatively time consuming analytic solution but are prone to efficient hardware implementation using discretized representations.

This paper is organized as follows. In Section 2 a survey of previous related work is given. In Section 3 we show that the C-space bitmap can be obtained by a convolution of the workspace and the robot. In Section 4 we discuss the conditions that make possible the use of the FFT algorithm for the efficient calculation of convolution. Then, in Section 5, we show how our problem can match these conditions and we give an algorithm for computing the C-space using the FFT. In Section 6 we present an experimental evaluation of the algorithm and in Section 7 we discuss the possibility of hardware and parallel implementations using recent work on the FFT.

2. Survey of Existing Algorithms

Let us first introduce some notation. The robot is denoted by $\mathcal{A}$. $\mathcal{A}$ is a rigid body and moves in the workspace $\mathcal{W} \subset \mathbf{R}^n$, with $n = 2$ or $n = 3$. Let $\mathcal{B}$ denote one of the obstacles in $\mathcal{W}$, and $\mathcal{C}$ the C-space of $\mathcal{A}$ [16]. The subset of $\mathcal{W}$ occupied by $\mathcal{A}$ at configuration q is denoted by $\mathcal{A}(q)$. That is $\mathcal{A}(q) = \mathcal{A} + q = \{x + q : x \in \mathcal{A}\}$. The obstacle $\mathcal{B}$ in $\mathcal{W}$ maps into $\mathcal{C}$ to the region $\mathcal{CB} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{B} \neq \emptyset\}$, which is called a C-obstacle.

Many proposed algorithms for computing the C-obstacles for $\mathcal{A}$ deal with the case where $\mathcal{A}$ can only translate and restrict the shape of the robot and the obstacles. We begin our survey with these algorithms.

The case where $\mathcal{A}$ and $\mathcal{B}$ are convex polygons has been studied in [15, 17] and an optimal algorithm has been proposed by Lozano-Pérez [17] and Guibas [6]. They obtain the vertices of the also convex $\mathcal{CB}$ in $O(n_{\mathcal{A}}+n_{\mathcal{B}})$ time, where $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ denote the number of vertices of $\mathcal{A}$ and $\mathcal{B}$ respectively. In the case where $\mathcal{A}$ and $\mathcal{B}$ are non-convex polygons, Sharir [24] gave an algorithm which computes the boundary of $\mathcal{CB}$ in $O(n_{\mathcal{A}}^2 n_{\mathcal{B}}^2 \log(n_{\mathcal{A}} n_{\mathcal{B}}))$ time. In the case where $\mathcal{A}$ and $\mathcal{B}$ are generalized convex polygons, i.e., regions bounded by straight line segments or circular arcs, Laumond [12] showed that $\mathcal{CB}$ can be computed in $O(n_{\mathcal{A}} + n_{\mathcal{B}})$ time. When $\mathcal{A}$ and $\mathcal{B}$ are locally non-convex generalized polygons a decomposition of $\mathcal{A}$ and $\mathcal{B}$ into convex generalized polygons yields an $O(n_{\mathcal{A}}^2 n_{\mathcal{B}}^2 \log(n_{\mathcal{A}} n_{\mathcal{B}}))$ algorithm [12]. For $\mathcal{A}$ or $\mathcal{B}$ not locally non-convex, the algorithm has $O((n_{\mathcal{A}} n_{\mathcal{B}}+c) \log(n_{\mathcal{A}} n_{\mathcal{B}}))$ time complexity, where c is $O(n_{\mathcal{A}}^2 n_{\mathcal{B}}^2)$ in the worst case [10]. The case where $\mathcal{A}$ and $\mathcal{B}$ are convex polyhedra has been studied in [16, 17]. The best known algorithm has been given by Guibas and Seidel [7]. It constructs the boundary of $\mathcal{CB}$ in $O(n_{\mathcal{A}} + n_{\mathcal{B}} + c)$ time, where c is in the worst case $O(n_{\mathcal{A}} n_{\mathcal{B}})$. Finally, the method of Avnaim and Boissonnat computes the boundary of $\mathcal{CB}$ in $O(n_{\mathcal{A}}^3 n_{\mathcal{B}}^3 \log(n_{\mathcal{A}} n_{\mathcal{B}}))$ time, when $\mathcal{A}$ and $\mathcal{B}$ are any polyhedra.[1]

The above algorithms *analytically* compute the boundary of the C-obstacle for each connected workspace obstacle and then take the union of all constructed boundaries to find the boundary of all C-obstacles. Path planners operating in discretized C-spaces feed the results produced by these methods to routines that build the C-

[1]This method is an adaptation of the algorithm in [1] which constructs an exact representation of a C-obstacle in the case where $\mathcal{A}$ can translate and rotate in the plane.

space bitmap. For example, Latombe [11] and Lengyel et al. [14] used the $O(n_{\mathcal{A}}+n_{\mathcal{B}})$ algorithm of Lozano-Pérez and Guibas to compute the vertices of the C-obstacles and then filled the C-obstacles with a raster graphics polygon-filling routine. Paths were computed in the produced bitmap.

Other methods for computing C-space maps that are closer to the spirit of this paper have been proposed. Newman and Branicky [19] identify 'elemental building blocks' (shapes) that are easily transformed from the workspace to the C-space. They store the C-space transforms of these shapes, frequently as bitmaps, and combine them to obtain the C-space bitmaps for complex shapes. They report experiments where the running time of their algorithm is very good. Lozano-Pérez and O'Donnell [18] implemented an algorithm on the Connection Machine that computes a family of C-space bitmaps which are then combined to construct bitmaps for more complex workspace objects and a six-degree-of-freedom robot. An algorithm given by Dehne, Hassenklover and Sack [5] computes C-space bitmaps for arbitrary obstacles but 'rectilinearly' convex robots on a $N \times N$ mesh of processors in $O(N)$ time. The above bound is generalized in our paper to non-convex robots. This improvement is also reported in similar work done independently by Schwarzkopf [23].

There are some inherent differences between the algorithms that analytically compute the C-obstacles and the algorithms that construct bitmaps for the C-space. The time complexity of the first is measured in terms of the number of vertices of the robot and the obstacles. On the other hand, the running time of the bitmap algorithms is a function of the size of the constructed bitmap. Typically, robots and obstacles with many vertices demand a high resolution of discretization for describing their shapes with sufficient accuracy. Hence, there is a relationship between the two complexity measures, but this relationship is not well-defined.

In the next sections we describe a method for computing C-space maps whose only inputs are a bitmap of the workspace and an algorithm to draw the robot on that bitmap. The output is a C-space bitmap representing both the boundary and the interior of the C-obstacles (1's) and the free space (0's). The running time of our method is independent of the number and the shape of the obstacles in the workspace $\mathcal{W}$, and depends only on the resolution of discretization.

3. Configuration Space as a Convolution

In this section, we explain in two dimensions how the C-space can be obtained as a convolution of the workspace and the translating robot. The same discussion also applies in three dimensions.

Consider a workspace $\mathcal{W} = [a, b] \times [c, d] \subset \mathbf{R}^2$. We discretize $\mathcal{W}$ into a $N \times N$ array W[2], where N is large enough to represent the obstacles in $\mathcal{W}$ with the desired accuracy. We define

$$cell_{i,j} = [a + i\frac{(b-a)}{N}, a + (i+1)\frac{(b-a)}{N}] \times [c + j\frac{(d-c)}{N}, c + (j+1)\frac{(d-c)}{N}]$$

where $i, j \in S = \{0, \ldots, N-1\}$, If there is an obstacle anywhere in the cell $cell_{i,j}$ we let $W(i,j) = 1$, else $W(i,j) = 0$. The bitmap array W can easily be constructed if the obstacles are input as a collection of algebraic shapes, e.g., polygons represented by their vertices. When the data about the obstacles is obtained through sensing, the bitmap representation may be more straightforward to obtain than an algebraic representation.

The C-space $\mathcal{C}$ is the set of all triples (x_r, y_r, θ_r), where (x_r, y_r) are the coordinates of a fixed reference point on the robot, $p_{\mathcal{A}}$, and θ_r is the orientation of the robot. In general the parameters x_r, y_r, θ_r can assume any real value in $[a, b] \times [c, d] \times [0, 2\pi]$. We discretize the workspace and the orientations of the robot and define $x = a + i\frac{(b-a)}{N}$, $y = c + j\frac{(d-c)}{N}$ and $\theta = k\frac{2\pi}{N}$, where $i, j, k \in S$. The C-space can then be stored as a $N \times N \times N$ binary array.[3]

The robot $\mathcal{A}$ can be approximated by a set of points (i, j), $i, j \in S$, which are drawn by a simple procedure given the orientation θ of the robot and the coordinates

[2] In general, $\mathcal{W}$ is discretized into a $N \times M$ array. The fact that we use a square array is only for simplifying our presentation. It has no particular importance in our method.

[3] Footnote 2 applies for θ as well.

(x, y) of $p_{\mathcal{A}}$. For each fixed value of (x, y, θ) we consider the $N \times N$ binary array $A_{(x,y,\theta)}$ where only the points that belong to the robot are marked with 1's.

With the conventions of the previous paragraphs, a point (x, y, θ) in the (discrete) C-space is legal (free) if and only if

$$C(x, y, \theta) \equiv \sum_{i,j=0}^{N-1} W(i, j) A_{(x,y,\theta)}(i, j) = 0.$$

We observe that whenever θ is fixed and x, y are varying, the various bitmaps $A_{(x,y,\theta)}$ are all translations of each other, and in particular of $A_{(0,0,\theta)}$. Then

$$C(x, y, \theta) = \sum_{i,j} W(i, j) A_{(0,0,\theta)}(i - x, j - y).$$

For the moment we will ignore any complication that might arise concerning the range of the indices i, j. We can assume that W and $A_{(0,0,\theta)}$ are infinite in all directions and padded with zeros (which indicate free space and do not affect the above sum). What we have shown is that the array $C(\cdot, \cdot, \theta)$ is the convolution of the arrays W and A'_θ, where

$$A'_\theta(i, j) = A_{(0,0,\theta)}(-i, -j).$$

Indeed, the convolution of two arrays Q and T is defined as $(Q \star T)(x, y) = \sum_{ij} Q(i, j) T(x - i, y - j)$. Hence, $C(\cdot, \cdot, \theta) = W \star A'_\theta$.

4. Computation of Convolution Via FFT

The convolution of two functions f and g can be computed by taking the Fourier Transform (FT) of the two functions, multiplying the two transforms pointwise, and then taking the inverse FT of the product. This is asserted by the Convolution Theorem given below:

Convolution Theorem: *If functions f and g defined on* $\mathbf{R}$ *are integrable then,* $\widehat{f \star g}(x) = \widehat{f}(x)\widehat{g}(x)$, *where* $\widehat{h}$ *denotes the* FT *of function* h.

However, calculating the convolution with the use of the Convolution Theorem is of computational interest only when the FFT algorithm can be applied. The FFT algorithm can be used when the functions f and g are periodic with the same period, or what amounts to the same thing, when they are defined on a 'cyclic' space. The convolution and the FT are then defined as follows for the two-dimensional case [21]:

Definition 1: *The convolution of two functions f and g defined on the set S^2, where $S = \{0, 1, \ldots, N-1\}$, is the function $(f \star g)(x, y) \equiv \sum_{i,j \in S} f(i, j) g(x-i, y-j)$ on S^2, where the arithmetic on the indices is done modulo N.*

Definition 2: *The* FT *of f on S^2 is the function $\hat{f}(x, y) = \sum_{j,k \in S} f(j, k) \zeta^{-jx-ky}$ on S^2, where $\zeta = \exp(2\pi i/N)$.*

The Convolution Theorem now holds for f and g defined as above. When this theorem is used, the convolution computed is the one given in Definition 1.

The classic FFT algorithm computes the one-dimensional FT of functions on S, as well as its inverse (which is again defined on S), in time $O(N \log N)$. The two-dimensional FT can be computed by first taking the one-dimensional FT of all rows of the array, which is a function on S^2, storing the results in the rows themselves, and then taking the FT of all columns and putting the result back in the columns. The time needed to compute a two-dimensional FT is $O(N^2 \log N)$. Analogous definitions hold for the k-dimensional FT whose computation takes $O(N^k \log N)$, when k is fixed.

5. Computation of Configuration Space

Basic Algorithm. Let us now consider the problem of computing the convolution of the workspace bitmap W and the robot map A'_θ. We could proceed by using the definition of the convolution which directly yields an $O(N^4)$ time procedure. Rather, we wish to use the FFT results and compute this convolution in $O(N^2 \log N)$ time. However, the functions W and A'_θ involved in the convolution are *not* 'cyclic' as is required by Definition 1. We eliminate this difficulty by simply setting $W(i, j)$ to 1 on the boundary of S^2. By doing so, we make sure that the robot cannot 'wrap around' the workspace without a collision.

The actual running time of the proposed method critically depends on how fast we can compute the discrete FFT of N points. The FFT algorithm is a very popular algorithm. Optimized implementations of this algorithm are available on virtually every computer. Many FFT implementations have been tailored to exploit the pipelines of RISC processors and achieve close to peak performance. As discussed in Section 7, FFT hardware can reduce the running time of our method.

Algorithm for a translating and rotating planar robot. In Figure 1 we give the algorithm that computes the C-space bitmap for a robot $\mathcal{A}$ that can translate and rotate in the plane. As before, x and y are the coordinates of a fixed reference point ($p_{\mathcal{A}}$) of the robot and θ is the orientation of the robot; N is the resolution of the discretization on the workspace bitmap W. If N is also the number of the desired orientations of the robot, a $N \times N \times N$ C-space bitmap CSPACE is constructed.[4]

If we have a good way to draw the robot in the bitmap for every new orientation, the running time of the algorithm is not significantly affected by the shape of the robot. Its time complexity is essentially $O(N^3 \log N)$. Robot drawing can often be simplified by drawing only the boundary of the robot. Then the region filled by 1's in the CSPACE bitmap is only a subset of the true C-obstacles, but it contains the boundary of the latter. For planners like the one described in [2], this is sufficient to prevent collision because the robot is never allowed to cross a C-obstacle boundary.

Algorithm for a translating robot in three dimensions Although we focused our previous discussion on a robot moving in two dimensions, the same method applies in three dimensions. If $\mathcal{A}$ can only translate, a three-dimensional C-space can be constructed in $O(N^3 \log N)$ time, by taking the three-dimensional convolution of the workspace W and the robot $A'_{\phi\theta\psi}$. If rotation is allowed to occur about a single axis, say θ, a four dimensional bitmap can be constructed by an algorithm similar to

[4]Even when the number of desirable orientations is chosen different from N, it is expected to be $O(N)$.

1 Put $W(i,j) = 1$ when either i or j is 0 or $N-1$,
 or an obstacle is present.
2 Compute $\widehat{W}$, the 2D FT of W.
3 For all desired values of θ:
4 begin
5 Construct A'_θ.
6 Compute $\widehat{A'_\theta}$, the 2D FT of A'_θ.
7 Let $X = \widehat{W} \cdot \widehat{A'_\theta}$ (pointwise multiplication).
8 Compute Y as the inverse FT of X.
9 Let CSPACE$(x, y, \theta) = 1$ if and only if $|Y(x,y)| = 1$.
10 end

Figure 1: Computation of C-space map for a rigid robot that can translate and rotate in the plane.

the one given in Figure 1. The algorithm then has a time complexity $O(N^4 \log N)$, if N is the number of desirable values of θ. The same principle applies if rotation is allowed along the other axes. However, if a reasonable resolution of discretization is desired, the size of the C-space bitmap becomes very large and it is not realistic to store it in the memory of current workstations.

6. Experimental Evaluation

We have implemented[5] the FFT-based algorithm of Figure 1. We have conducted a series of experiments aimed at comparing its performance with (i) an implementation of the $O(n_{\mathcal{A}} + n_{\mathcal{B}})$-time algorithm of Lozano-Pérez [17] and Guibas [6] and

[5] Our implementation follows the description in Figure 1, except for a minor change to compensate for possible arithmetic errors. We set CSPACE$(x, y, \theta) = 1$ when $|Y(x,y)| >$ THRESHOLD, instead of when $|Y(x,y)| = 1$ (line 9). In our experiments THRESHOLD was set to 0.9.

(ii) an optimized implementation of the $O(N^4)$-time algorithm that computes the convolution of the workspace and the robot in a straightforward way. All algorithms in this section are implemented in C and run on a DEC 5000 workstation.

Comparison with the linear $O(n_{\mathcal{A}} + n_{\mathcal{B}})$ algorithm. As discussed in Section 2, the input and output data are different for algorithms that compute C-obstacles (in this case the $O(n_{\mathcal{A}} + n_{\mathcal{B}})$ algorithm) and algorithms that compute C-space bitmaps (in this case the FFT-based algorithm). An analytic comparison between these algorithms does not seem possible. We tried to establish a reasonable framework for an experimental comparison. Our aim is to provide orders of magnitude for the running times of the FFT-based algorithm and the $O(n_{\mathcal{A}} + n_{\mathcal{B}})$ algorithm, which has been used in many motion planners [11, 14].

The input of the implemented FFT-based algorithm is an $N \times N$ bitmap W of the workspace. The robot is defined as a polygon. Its bitmap representation is not precomputed; the algorithm computes the bitmap representation of the robot's boundary for each of the N orientations of the robot that we considered. The output is an $N \times N \times N$ bitmap CSPACE representing the C-space.

The input of the implemented linear algorithm is a collection of convex polygons representing the robot and another collection of polygons representing the workspace obstacles. Hence, the decomposition of non-convex polygons into convex ones is not part of the running times given below. The output of the linear algorithm is entered into a routine mapping this boundary in a C-space bitmap of the same resolution as the one built by the FFT-based algorithm. This bitmap is the output of the algorithm. When we have an analytic description of the C-obstacles, we can construct a CSPACE bitmap of high resolution with accuracy. In the FFT-based algorithm, we first project the obstacles and the robot in the workspace bitmap and then compute a CSPACE bitmap of the same resolution. Thus, in order to obtain an accurate C-space bitmap, we have to work with a resolution high enough to represent the shape of the obstacles with accuracy. In our experiments we made sure that the CSPACE bitmaps produced by the algorithms we compare were almost

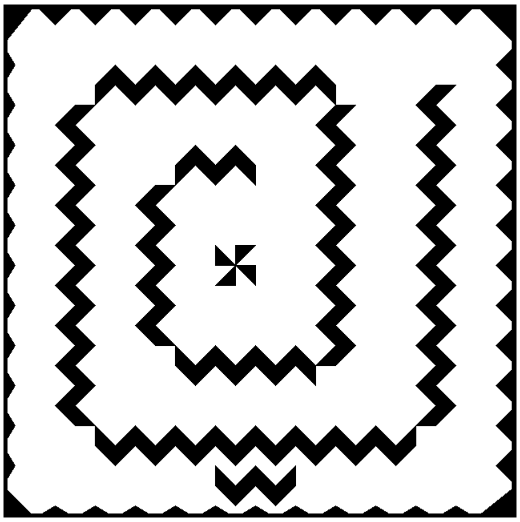

Figure 2: Two-dimensional workspace with 150 convex polygon obstacles

the same.

Figure 2 shows the workspace (150 convex polygonal obstacles) and Figure 3 shows the three robots we used in one series of experiments. The larger dimension of the robots is approximately 1/10 of the dimension of the workspace. The workspace bitmap W was represented as a 128×128 array and we considered 128 orientations for the robot. Hence, the algorithms built a $128 \times 128 \times 128$ bitmap CSPACE.

- For the convex robot of Figure 3(a), the FFT-based algorithm took 90 seconds to compute CSPACE and the linear algorithm 11 seconds.
- For the robot of Figure 3(b), which is non-convex, the FFT-based algorithm took 91 seconds and the linear algorithm 22 seconds.
- For the robot shown in Figure 3(c), which has four convex parts, the FFT-based algorithm took 91 seconds and the linear algorithm 32 seconds.

Experiments show clearly that the linear algorithm is preferable for simple polygonal robots and workspaces. However, as the complexity of the environment increases, our FFT-based algorithm becomes more and more comparable. One could

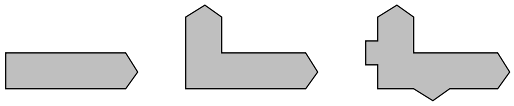

Figure 3: Polygonal robots

Size of robot bitmap	Bitwise operations (sec)
5×5	20
10×10	39
20×20	77
30×30	115
40×40	199

Table 1: Direct convolution timings with a 128×128 workspace

imagine more complex environments where the gap in the running times of the FFT-based and the linear algorithm will be reduced or even inverted.

We emphasize that we ran our experiments with algorithms implemented in software on a conventional single-processor architecture. A hardware or parallel implementation of the FFT algorithm would certainly lower the 'break-even' complexity where the FFT-based algorithm becomes preferable to the linear algorithm.

Comparison with the direct $O(N^4)$ convolution algorithm. Although the FFT-based algorithm has an asymptotically better running time then the direct algorithm, the latter requires only a small bitmap for the robot and can be implemented using (fast) bitwise operations, instead of (slow) multiplications.

The input and the output of the implemented direct convolution algorithm are exactly the same as those of the FFT-based algorithm. The direct algorithm augments the workspace bitmap and pads it with zeros to avoid problems at boundary configurations when the convolution is computed. Table 1 summarizes experimental results for a 128×128 workspace. The FFT-based algorithm computes the bitmap CSPACE in approximately 90 seconds. Column 2 of Table 1 shows the time required to compute CSPACE for different sizes of the bitmap of the robot when bitwise operations are used. It is clear that the direct approach is preferable when the size of the robot is small. Bitwise operations are implemented in hardware in the machine we used and take at most a couple of machine cycles.

7. Hardware and Parallel Implementations

A hardware implementation of the basic FFT-based method for computing C-space maps is possible because of the uniformity of the calculations involved in the algorithm. Such an implementation may substantially reduce the running time of the algorithm. Even if the whole method is not implemented in hardware, specialized FFT hardware [3] can be used. The latter is widely available and has been used for years in signal processing and other applications.

Hardware implementations are also possible for the direct convolution algorithm and cannot be regarded as an advantage of the FFT-based algorithm only. This issue has been studied in the context of the morphological operations used in image processing [8]. The computation of these operations requires a small bitmap, referred to as a *kernel*, to be moved over the original image. Many specialized architectures have been proposed [22, 20] to efficiently perform these operations. However, these architectures achieve high performance for operations with large images and small kernels. Typical kernel sizes for a 512×512 image are 3×3 or 5×5. In our problem, we might not want to restrict ourselves to such small robots.[6]

As far as our basic algorithm is concerned, its parallel implementations can be based on existing parallel implementations of the FFT. Let us mention some existing results. Leighton [13] showed that the N-point discrete FFT can be implemented in $\log N$ steps on an $N(\log N + 1)$-node butterfly or an N-node hypercube. It can also be implemented in $O(\log N)$ steps on an N-node butterfly [13]. Issues related to

[6] Even in AVG applications, path planning frequently has to be done locally, since more global planning often makes use of higher-level symbolic maps defining intermediate goals.

the implementation of the FFT on hypercubes are discussed in [25]. On an $N \times N$ mesh of processors, it is easy to implement the (two-dimensional) FT of a function defined on N^2 points in $O(N)$ time [13]. Then our basic algorithm will itself have a complexity of $O(N)$. This is asymptotically optimal for the problem, and is an improvement over the algorithm of Dehne, Hassenklover and Sack [5] which can only deal with 'rectilinear' robots. For the Connection Machine there exist many implementations of the FFT [9, 26]. On a shared memory bus-based multiprocessor with few processors (8) an implementation of the multidimensional FFT exhibits a speedup close to linear with the number of processors used.[7]

8. Conclusion

We presented an FFT-based algorithm to compute the C-space map used by several path planners. The algorithm is based on the observation that the C-space map is the convolution of the workspace and the robot and applies to obstacles and robots of any shape input as bitmaps. The FFT-based method produces a bitmap representation of the C-space that is directly exploitable by potential field based path planners such as those described in [2, 14]. In particular, this bitmap allows collision checking to be done in $O(1)$ time for a rigid robot and in $O(K)$ time for a robot with K rigid parts. For a fixed discretization, the running time of our algorithm is independent of the number and the shape of the obstacles in the workspace.

The FFT-based method was presented in the case of a planar translating robot. Rotation is not directly handled by the method and requires slice by slice construction of the C-space bitmap. The method also applies to a three-dimensional robot that can only translate. Furthermore, it can be used when the robot is a planar articulated linkage, by building a three-dimensional C-space for each link.

[7]The almost linear speedup is justified by the way the k-dimensional FFT is computed. For example, for k=2, we first take the FFT of the rows and then the FFT of the columns. We can assign different rows/columns to different processors. The partial computations are independent.

Our method depends highly on our ability to perform the FFT. When implemented in software on a single-processor computer, its running time can be a disadvantage, especially when the obstacles and the robot are simple. However, when the combined complexity of the obstacles and the robot is high, the FFT-based method becomes advantageous. It can also benefit from existing special-purpose hardware. Implementations are possible on a wide variety of parallel architectures, by exploiting recent work done on parallel implementations of the FFT. The use of specific hardware or parallel architectures can drastically reduce the running time of our algorithm, making it compare even more favorably to previous algorithms for computing C-space maps.

Acknowledgments: This research was partially funded by DARPA contract DAAA21-89-C0002. An earlier version of this work appeared in the IEEE Conference on Robotics and Automation, 1993. The author thanks T. Lastennet for providing the code of the linear-time algorithm for computing C-space maps and J.-C. Latombe for his valuable comments and reading of this paper. Finally, the comments of the anonymous reviewers of this paper were very helpful.

References

[1] F. Avnaim, J.D. Boissonnat, *Polygon Placement Under Translation and Rotation*, Technical Report No. 890, INRIA, 1988.

[2] J. Barraquand, J.C. Latombe, "Robot Motion Planning: A Distributed Representation Approach," *The Int. J. of Robotics Research*, The MIT Press, Cambridge, MA, 10(6), December 1991, 628–649.

[3] G. Bilardi, S. Hornick, M. Sarrafradeh, "Optimal VLSI Architectures for Multidimensional DFT," *ACM Comp. Architecture News*, 19(1), March 1991, 45–52.

[4] R.A. Brooks, T. Lozano-Pérez, "A Subdivision Algorithm in Configuration Space for Findpath with Rotation," *IEEE Tr. on Syst., Man, and Cyber.*, 15(2),

1985, 224–233.

[5] F. Dehne, A.L. Hassenklover, J.R. Sack, "Computing the configuration space for a robot on a mesh-of-processors," *Parallel Comput.*, 12(2), 1989, 221–231.

[6] L. Guibas, L. Ramshaw, J. Stolfi, "A Kinetic Framework for Computational Geometry," *Proc. of the IEEE Symp. on Found. of Computer Science* , 1983, 100–111.

[7] L. Guibas, R. Seidel, "Computing Convolution by Reciprocal Search," *Proc. of the ACM Symp. on Computational Geometry* , Yorktown Heights, NY, 1986, 90–99.

[8] R. Haralick, L. Shapiro, *Computer and Robot Vision*, Addison Wesley, 1992.

[9] L. Johnsson, R. Krawitz, "Cooley-Tukey FFT on the Connection Machine", *Parallel Computing* (18), 1992, 1201–1221.

[10] J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[11] J.C.Latombe, "A Fast Path Planner for a Car-Like Indoor Mobile Robot," *Proc. of the 9th Nat. Conf. on Artificial Intelligence*, 1991, 659–665.

[12] J.P. Laumond, "Obstacle Growing in a Non-Polygonal World," *Information Processing Letters*, 25(1), 1987, 41–50.

[13] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: arrays, trees, hypercubes*, Morgan Kaufmann Publishers Inc., Los Altos, CA, 1992.

[14] J. Lengyel, M. Reichert, B.R. Donald, D. P. Greenberg, "Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware," *Proc. of SIGGRAPH'90*, Dallas, TX, 1990, 327–335.

[15] T. Lozano-Pérez, M. Wesley, "An algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, 22(10), Oct 1979, 560–570.

[16] T. Lozano-Pérez, "Automatic Planning of Manipulator Transfer Movements," *IEEE Tr. on Systems, Man, and Cybernetics*, 11(10), 1981, 681–698.

[17] T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach," *IEEE Tr. on Computers*, 32, 1983, 108–120.

[18] T. Lozano-Pérez, P. O'Donnell, "Parallel Robot Mation Planning", *Proc. of the IEEE Int. Conf. on Robotics and Automation*, April 1991, 1000–1007.

[19] W. Newman, M. Branicky, "Real-Time Configuration Space Transforms for Obstacle Avoidance," *The Int. J. of Robotics Research*, 10(6), 1991, 650–667.

[20] W. Pratt, "A Pipeline Architecture for Image Processing and Analysis," *IEEE Comp. Arch. for Pat. Anal. and Image Database Management*, Miami, 1985, 516–520.

[21] R. Ramirez, *The FFT Fundamentals and Concepts*, Prentice Hall, Englewood Cliffs, NJ, 1985.

[22] A.P. Reeves, "Parallel Computer Architectures for Image Processing," *Computer Vision, Graphics and Image Processing*, 25(1), 1984, 68–88.

[23] O. Schwarzkopf, "Computing Convolutions on Mesh-Like Structures", Department of Computer Science, Utrecht University, 3508 TB Utrecht, The Netherlands, manuscript.

[24] M. Sharir, "Efficient Algorithms for Planning Purely Translational Collision-Free Motion in Two and Three Dimensions," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1987, 1326–1331.

[25] P. Swarztrauber, "Multiprocessor FFTs," *Parallel Computing*, 5(1), 1987, 197–210.

[26] Ch. Tong, P. Swarztrauber, "Ordered FFTs on a Massively Parallel Hypercube Multiprocessor," *Parallel Computing*, 1991, 50–59.

[27] D. Zhu, J.C. Latombe, "New Heuristic Algorithms for Efficient Hierarchical Path Planning," *IEEE Tr. on Robotics and Automation*, 7(1), 1991, 9–20.