Fig. 6.   Solutions found by the novel approach of inverse kinematic functions (Box position {0.3, −0.6, 0.2}).

generalized inverse kinematic functions obtained are used in order to plan the trajectory of a manipulator in approach and catching operations, where the object is not initially located inside the robot's workspace. More details on this application can be found in [10].

## ACKNOWLEDGMENT

## REFERENCES

[1]   D. L. Pieper, "The kinematics of manipulators under computer control," Ph.D. dissertation, Stanford Artificial Intelligence Laboratory, Stanford University, 1968.

[2]   J. Angeles, "On the numerical solution of the inverse kinematic problem," *Int. J. Robot. Res.*, vol. 4, No. 2, pp. 21–37, 1985.

[3]   K. C. Gupta and K. Kazerounian, "Improved numerical solutions of inverse kinematics of robots," in *Proc., IEEE Int. Conf. Robot. and Automat.*, 1985.

[4]   M. Takano, "A new effective solution for inverse kinematic problem (synthesis) of a robot with any type of configuration," *J. Faculty of Eng. University of Tokyo(B)*, vol. 38, no. 2, pp. 107–135, 1985.

[5]   L. W. Tsai and A. P. Morgan, "Solving the kinematics of the most general six and five-degree-of-freedom manipulator by continuation methods," *ASME J. Mechanisms, Transmissions and Automation in Design*, vol. 107, pp. 189–200, 1985.

[6]   M. Raghavan and B. Roth, "Kinematic analysis of the 6R manipulator of general geometry," in *Int. Symp. Robot. Res.*, 1989, pp. 314–320.

[7]   D. Manocha and J. Canny, "Real time inverse kinematics for general 6R manipulators," in *Proc. IEEE Conf. on Robot. and Automat.*, 1992, vol. 1, pp. 383–389.

[8]   E. D. Pohl and H. Lipkin, "A new method of robotic rate control near singularities," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 1991, pp. 1708–1713.

[9]   ——, "Kinematics of complex joint angles in robotics," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 1990, pp. 86–91.

[10]  J. Côté, C. Gosselin and D. Laurendeau, "A graphic simulator for the study of robotic tracking and catching operations," in *Proc. ASME Comput. in Eng.*, 1992, vol. 1, pp. 141–146.

[11]  ——, "Tracking a moving object with a 6-DOF manipulator," in *Proc. SPIE Applicat. of Artificial Intell. 1993: Machine Vision and Robot.*, 1993, vol. 1964, pp. 300–311.

[12]  C. Gosselin, J. Côté and D. Laurendeau, "Inverse kinematic functions for approach and catching operations," *IEEE Trans. Syst., Man and Cybern.*, vol. 23, no. 3, pp. 783–791, 1993.

[13]  M. W. Spong and M. Vidyasagar, Robot Dynamics and Control.   New York: Wiley, 1989, p. 336.

# Computation of Configuration-Space Obstacles Using the Fast Fourier Transform

Lydia E. Kavraki

*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.

## I. 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.

This paper is organized as follows. Section II gives a survey of previous related work. In Section III we show that the C-space bitmap is a convolution of the workspace and the robot. In Sections IV and V we discuss how to use the FFT algorithm for the efficient calculation of the above convolution. In Section VI we present an experimental evaluation of the proposed algorithm and in Section VII we discuss the possibility of hardware and parallel implementations using recent work on the FFT.

## II. 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 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 $O(n_A + n_B)$ time, where $n_A$ and $n_B$ denote the number of vertices of $\mathcal{A}$ and $\mathcal{B}$, respectively. In the case where $\mathcal{A}$ and $\mathcal{B}$ are nonconvex polygons, Sharir [24] gave an algorithm which computes the boundary of $\mathcal{CB}$ in $O(n_A^2 n_B^2 \log(n_A n_B))$ 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 $O(n_A + n_B)$ time. When $\mathcal{A}$ and $\mathcal{B}$ are locally nonconvex generalized polygons a decomposition of $\mathcal{A}$ and $\mathcal{B}$ into convex generalized polygons yields an $O(n_A^2 n_B^2 \log(n_A n_B))$ algorithm [12]. For $\mathcal{A}$ or $\mathcal{B}$ not locally nonconvex, the algorithm has $O((n_A n_B + c) \log(n_A n_B))$ time complexity, where $c$ is $O(n_A^2 n_B^2)$ 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 $O(n_A + n_B + c)$ time, where $c$ is in the worst case $O(n_A n_B)$. Finally, the method of Avnaim and Boissonnat in [1] can be adapted to compute the boundary of $\mathcal{CB}$ in $O(n_A^3 n_B^3 \log(n_A n_B))$ time, when $\mathcal{A}$ and $\mathcal{B}$ are any polyhedra [10].

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-space bitmap. For example, Latombe [11] and Lengyel et al. [14] used the $O(n_A + n_B)$ 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 nonconvex robots. This improvement is also reported in similar to ours 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.

## III. 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 $W^1$, 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 $(x_r, y_r, \theta_r)$, where $(x_r, y_r)$ are the coordinates of a fixed reference point on the robot, $p_\mathcal{A}$, and $\theta_r$ is the orientation of the robot. In general the parameters $x_r, y_r, \theta_r$ 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.[2]

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 $\theta$ of the robot and the coordinates $(x, y)$ of $p_\mathcal{A}$. For each fixed value of $(x, y, \theta)$ 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 $(x, y, \theta)$ 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 $\theta$ 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 $A'_\theta$, 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, $C(\cdot, \cdot, \theta) = W \star A'_\theta$.

## IV. 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 $\widehat{f \star g}(x) = \hat{f}(x)\hat{g}(x)$, where $\hat{h}$ 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

---

[1] 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.

[2] Footnote 1 applies for $\theta$ as well.

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.

## V. COMPUTATION OF CONFIGURATION SPACE

*1) Basic Algorithm.* Let us now consider the problem of computing the convolution of the workspace bitmap $W$ and the robot map $A'_\theta$. 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 $A'_\theta$ 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 VII, FFT hardware can reduce the running time of our method.

*2) Algorithm for a translating and rotating planar robot.* In Fig. 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 $\theta$ 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.[3]

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

---

[3] Even when the number of desired 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 $\theta$:

4     begin

5         Construct $A'_\theta$.

6         Compute $\widehat{A'_\theta}$, the 2D FT of $A'_\theta$.

7         Let $X = \widehat{W} \cdot \widehat{A'_\theta}$ (pointwise multiplication).

8         Compute $Y$ as the inverse FT of $X$.

9         Let $\text{CSPACE}(x, y, \theta) = 1$ iff $|Y(x,y)| = 1$.

10    end

Fig. 1.  Computation of C-space map.

in [2], this is sufficient to prevent collision because the robot is never allowed to cross a C-obstacle boundary.

*3) 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 $\theta$, a four dimensional bitmap can be constructed by an algorithm similar to the one given in Fig. 1. The algorithm then has a time complexity $O(N^4 \log N)$, if $N$ is the number of desirable values of $\theta$. 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.

## VI. EXPERIMENTAL EVALUATION

We have implemented[4] the FFT-based algorithm of Fig. 1. We have conducted a series of experiments aimed at comparing its performance with (1) an implementation of the $O(n_\mathcal{A} + n_B)$-time algorithm of Lozano-Pérez [17] and Guibas [6] and (2) 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.

### A. Comparison with the Linear $O(n_\mathcal{A} + n_B)$ Algorithm

As discussed in Section II, the input and output data are different for algorithms that compute C-obstacles (in this case the $O(n_\mathcal{A} + n_B)$ 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 $O(n_\mathcal{A} + n_B)$ 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

[4] Our implementation follows the description in Fig. 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.



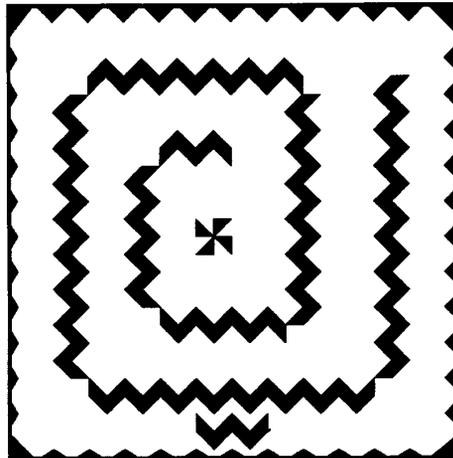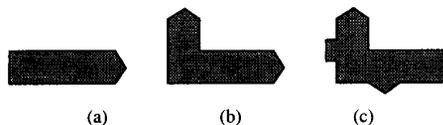Fig. 2.  Workspace with 150 convex polygon obstacles.



    (a)         (b)         (c)

Fig. 3.  Polygonal robots.

$N$ 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 nonconvex 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 the same.

Fig. 2 shows the workspace (150 convex polygonal obstacles) and Fig. 3 shows the three robots we used in one series of experiments. The largest dimension of the robots is approximately 1/10 of the dimension of the workspace. The workspace bitmap $W$ was represented as a $128 \times 128$ 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 Fig. 3(a), the FFT-based algorithm took 90 seconds to compute CSPACE and the linear algorithm 11 seconds.
- For the robot of Fig. 3(b), which is nonconvex, the FFT-based algorithm took 91 seconds and the linear algorithm 22 seconds.
- For the robot shown in Fig. 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

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

Fig. 4.   Direct convolution with a 128 × 128 workspace.

of the environment increases, our FFT-based algorithm becomes more and more comparable. One could 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.

### B. Comparison with the Direct Convolution Algorithm

Although the FFT-based algorithm has an asymptotically better running time than the $O(N^4)$ 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. Fig. 4 summarizes experimental results for a 128 × 128 workspace. The FFT-based algorithm computes the bitmap CSPACE in approximately 90 seconds. Column 2 of the table in Fig. 4 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.

### VII. 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.[5]

As far as our basic algorithm is concerned, its parallel implementations can be based on existing parallel implementations of the FFT.

[5] 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.

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 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.[6]

### VIII. 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.

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.

### REFERENCES

[1]  F. Avnaim and J. D. Boissonnat, "Polygon placement under translation and rotation," Technical Report No. 890, INRIA, 1988.

[6] 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.

[2] J. Barraquand and J. C. Latombe, "Robot motion planning: A distributed representation approach," *Int. J. Robot. Res.*, vol. 10, no. 6, pp. 628–649, 1991.

[3] G. Bilardi, S. Hornick, and M. Sarrafradeh, "Optimal VLSI architectures for multidimensional DFT," *ACM Comp. Architecture News*, vol. 19, no. 1, pp. 45–52, Mar. 1991.

[4] R. A. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Trans. Syst., Man, and Cybern.*, vol. 15, no. 2, pp. 224–233, 1985.

[5] F. Dehne, A. L. Hassenklover, and J. R. Sack, "Computing the configuration space for a robot on a mesh-of-processors," *Parallel Comput.*, vol. 12, no. 2, pp. 221–231, 1989.

[6] L. Guibas, L. Ramshaw, and J. Stolfi, "A kinetic framework for computational geometry," in *Proc. IEEE Symp. Found. of Comput. Sci.* , 1983, pp. 100–111.

[7] L. Guibas and R. Seidel, "Computing convolution by reciprocal search," in *Proc. ACM Symp. on Computational Geometry* , 1986, pp. 90–99.

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

[9] L. Johnsson and R. Krawitz, "Cooley-Tukey FFT on the connection machine," *Parallel Computing*, vol. 18, pp. 1201–1221, 1992.

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

[11] J. C. Latombe, "A fast path planner for a car-like indoor mobile robot," in *Proc. 9th Nat. Conf. Artificial Intell.*, pp. 659–665, 1991.

[12] J. P. Laumond, "Obstacle growing in a non-polygonal world," *Info. Proc. Lett.*, vol. 25, no. 1, pp. 41–50, 1987.

[13] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Los Angeles, CA: Morgan Kaufmann, 1992.

[14] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware," in *Proc. SIGGRAPH'90*, 1990, pp. 327–335.

[15] T. Lozano-Pérez and M. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communicat. ACM*, vol. 22, no. 10, pp. 560–570, Oct 1979.

[16] T. Lozano-Pérez, "Automatic planning of manipulator transfer movements," *IEEE Trans. Syst., Man, and Cybern.*, vol. 11, no. 10, pp. 681–698, 1981.

[17] T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach," *IEEE Trans. Comput.*, vol. 32, pp. 108–120, 1983.

[18] T. Lozano-Pérez and P. O'Donnell, "Parallel robot motion planning," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 1000–1007, Apr. 1991.

[19] W. Newman and M. Branicky, "Real-time configuration space transforms for obstacle avoidance," *Int. J. Robot. Res.*, vol. 10, no. 6, pp. 650–667, 1991.

[20] W. Pratt, "A pipeline architecture for image processing and analysis," in *IEEE Comp. Arch. for Patt. Anal. and Image Database Management*, 1985, pp. 516–520.

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

[22] A. P. Reeves, "Parallel computer architectures for image processing," *Comput. Vision, Graphics and Image Processing*, vol. 25, no. 1, pp. 68–88, 1984.

[23] O. Schwarzkopf, "Computing convolutions on mesh-like structures," in *Proc. 7th Int. Parallel Processing Symp.*, 1993, pp. 1083–1087.

[24] M. Sharir, "Efficient algorithms for planning purely translational collision-free motion in two and three dimensions," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1987, pp. 1326–1331.

[25] P. Swarztrauber, "Multiprocessor FFT's," *Parallel Computing*, vol. 5, no. 1, pp. 197–210, 1987.

[26] C. Tong and P. Swarztrauber, "Ordered FFT's on a massively parallel hypercube multiprocessor," *Parallel Computing*, pp. 50–59, 1991.

[27] D. Zhu and J.C. Latombe, "New heuristic algorithms for efficient hierarchical path planning," *IEEE Trans. Robot. and Automat.*, vol. 7, no. 1, pp. 9–20, 1991.

# Graph-Theoretic Deadlock Detection and Resolution for Flexible Manufacturing Systems

Hyuenbo Cho, T. K. Kumaran, and Richard A. Wysk

*Abstract*—Flexible manufacturing systems are capable of producing a broad variety of products and changing their characteristics quickly and frequently. This flexibility provides for more efficient use of resources, but makes control of these systems more difficult. Control problems previously unstudied now require practical resolution, like system deadlock. A system deadlock is a situation that arises due to resource sharing in manufacturing systems, when the flow of parts is permanently inhibited and/or operations on parts cannot be performed. This problem has been ignored by most scheduling and control studies, which usually assume infinite machine queue capacity and unlimited tooling resources. FMS's, however, have little or no queue capacity and limited tooling resources. In this paper, graph-theoretic deadlock detection and resolution procedures are presented which are suitable for real-time control of manufacturing systems. These procedures determine whether part movement in the system causes system deadlock or not. To this end, a system status graph representing part routings is virtually updated for every part movement before parts move physically to the next destination. Two types of system deadlocks, part flow deadlock and impending part flow deadlock, are detected using the updated system status graph. If a deadlock detection and recovery method is used to recover from a deadlock using a storage buffer, only part flow deadlocks need to be detected. On the other hand, if no buffer is available, both types of existing as well as impending system deadlocks need to be detected to avoid a deadlock situation.

## I. INTRODUCTION

A flexible manufacturing system (FMS) can be defined as an integrated system of equipment and processes through a network of computers that is capable of producing a variety of product economically [1]. FMS's have provided the necessary flexibility to enable manufacturers to cope with the increased demands of low and medium volume production. Furthermore, FMS's capture the dynamic nature of manufacturing processes due to the concurrent flow of various parts as well as sharing of different types of resources. Even though these characteristics increase system efficiency and productivity, they bear practical, but undesirable problems related to control, e.g., system deadlock. A system deadlock is a situation that arises due to resource sharing in unmanned manufacturing systems, where the flow of parts is permanently inhibited and/or operations on parts cannot be performed.

The occurrence of system deadlocks, unlike blocking, stalls activities in part of or the entire system and makes part flow impossible. As a result, system deadlock has become a critical scheduling and control problem [2]-[5]. This problem has been ignored by most scheduling and control studies which usually assume infinite queue capacity. For instance in classical simulation models, parts balk after the storage buffers become full. For most practical FMS applications, this situation is not true.

A *part flow deadlock* is a situation in which transition(s) of a part are impossible, e.g., some or all parts can no longer move in a manufacturing system, because parts have been allocated in a