Configuration Space of 3D Mobile Robots: Parallel Processing

R. Theron theron@usal.es

V. Moreno vmoreno@abedul.usal.es

B. Curto bcurto@abedul.usal.es

F. J. Blanco jblanco@abedul.usal.es

Departamento de Informática y Automática
Universidad de Salamanca
Salamanca, Spain

Abstract

In this paper we present an analysis that indentifies opportunities of introducing parallelism in the calculation of the configuration space. When the Convolution Theorem is used for this evaluation, two levels have appeared within the basic operation of the algorithms and another one for the employed computational tool (Fast Fourier Transform, FFT).

The method has been applied to robots moving on a plane in a 3D workspace. As a result, an optimal reduction of the calculation time is obtained: the parallel implementation, working on a tetra-processor, is more than three times faster than the sequential one. Finally, it can be added that the proposed design for the parallel algorithm is general and easily implementable for articulated robots.

1 Introduction

Currently, the field of robotics is pursuing the creation of autonomous robots. To achieve this objective, it is indispensable to process a set of tasks [9] with different degrees of knowledge. In order to face the strong computational load derived from the execution of all the invloved procedures, the utilization of distributed processing techniques is needed [1].

One of the intelligent tasks to be carried out is *path planning* [9], that looks for a route which permits the robot to move from an initial configuration to the desired one, without collisions with the objects found inside its workspace. Among the different existing techniques, *global planning* appears, that considers a total knowledge of the environment or workspace of the robot. It is simpler to obtain these paths in the configuration space (*C-space*) than in the workspace of the robot, since in the first one the position and orientation of the robot can be defined by a single point [10]. The present work will be centered in the seek of the *free C-space*, that later will be used for planning robot movements. Or the opposite, configurations that produce a collision will be called *C-obstacle*. The evaluation of the C-space supposes a problem with a massive computational cost; furthermore, it has strong calculation time restrictions. This results in the necessity of incorporating advanced processing techniques [7].

In spite of the significant advances in the processing speed, sequential processors are very far from providing sufficient computing capacity for the intelligent robotic systems. On the other hand, modern VLSI technology provides an unique opportunity to tighten this separation by means of parallel processing [2].

Among all the parallel paradigms, message-passing upon point-to-point connections appears as a powerful valid solution for both shared memory systems and distributed memory systems [4], and is one of the most commonly used.

Previous works ([3], [11]) have demostrated how the intratability of the C-space problem would benefit from the use of parallel and distributed computing techniques. In this paper, it is the calculation of the C-space itself that is studied in order to find the best design and implementation of the algorithm.

The remainder of this paper has been divided into the following sections: In the second one, a general method for the evaluation of the configuration space of robots is analyzed looking for its inherent parallelism. In the third section, a sequential algorithm for the computing of the robot C-space is presented for a given case study. Next, the fourth section shows the parallel implementation with detailed master and slaves functions. To finalize, the main results and conclusions are presented.

2 Parallel C-space Evaluation

In this section, we are going to present the mathematical formalism that was used for the computation of the C-space. Next, the opportunities for introducing concurrent calculus will be studied.

2.1 Mathematical formalism

In [5] a mathematical formalism for the C-obstacle evaluation is proposed where two functions are defined: one describing the robot, A; and another one describing the obstcales, B.

Let W and C be the workspace and the configuration space of a robot, respectively.

Definition 1. Let $\mathbf{A}(q)$ be the set of points of W that represents the robot A at configuration q.

The function $A : C \times W \rightarrow [0, 1]$ is defined by

$$A(q,x) = \begin{cases} 1 & \text{if } x \in \mathbf{A}(q) \\ 0 & \text{if } x \notin \mathbf{A}(q) \end{cases} \tag{1}$$

Definition 2. Let $\mathbf{B}$ be the subset of W constituted by the obstacles. The function $B : W \rightarrow [0, 1]$ is defined by

$$B(x) = \begin{cases} 1 & \text{if } x \in \mathbf{B} \\ 0 & \text{if } x \notin \mathbf{B} \end{cases} \tag{2}$$

Definition 3. Let $CB : C \rightarrow R$ be the function defined by

$$CB(q) = \int A(q,x)B(x)dx \quad \forall q \in C \tag{3}$$

The C-obstacle region $\mathbf{CB_f}$ is defined as

$$\mathbf{CB_f} = \{q \in C / CB(q) > 0\}$$

In order to know whether the robot A at a given configuration q collides with the obstacles or not, it is necessary to evaluate (3). A point x of W will be expressed by the coordinates $(x_1, \cdots, x_n)$, where n is the dimension of W. A configuration q will be represented by $(q_1, \cdots, q_m)$, that specify the position and orientation of the robot. So, $CB(q_1, \cdot, q_m)$ function can be calculated as,

$$\int A(q_1, \cdot, q_m, x_1, \cdot, x_n)B(x_1, \cdot, x_n)dx_1, \cdot, dx_n \tag{4}$$

It is difficult to compute the previous integral; if the adequate coordinate funtions in W and C are chosen, this expression can be calculated in a simpler way [5], by means of a convolution product of two functions. Thus, if A' has a lighter functional dependency, being the robot $\mathbf{A}$ independent of a subset of parameters $(q_1, \cdot, q_r)$, when a relationship with some coordinate functions $(x_1, \cdot, x_r)$ is found, the convolution product appears. Now, thanks to the Convolution theorem, the evaluation of the integral can be done using the Fourier transform. Then, $\mathcal{F}[CB]$ will be calculated as

$$\int \mathcal{F}\left[A'_{(0,\cdot,0,q_{r+1},\cdot,q_m)}(q_1, \cdot, q_r, x_{r+1}, \cdot, x_n)\right]_{(x_1,\cdot,x_r)} \mathcal{F}\left[B(q_1, \cdot, q_r, x_{r+1}, \cdot, x_n)\right]_{(x_1,\cdot,x_r)} dx_{r+1} \cdot dx_n \tag{5}$$

2.2 Going Parallel

In expression (4), the opportunity of carrying out calculations in parallel was already observed: the execution of N^m integrals of dimension n will be necessary, being N the chosen discretization. But, in expression (5), the calculation is reduced to the evaluation of N^{m-r} operations. On the other hand, the integration, that is performed for the spatial coordinates where there is no convolution $(x_{r+1}, \cdots, x_n)$, provides a new opportunity of parallelization.

Thus, in the integral (5) three levels of parallelism can be found:

- **Fourier Transform level:** the well known Fast Fourier Transform (FFT) algorithm, which is inherently parallel is the computational tool to calculate this level.

- **Spatial variable level:** for those variables where the convolution can't be found $(x_{r+1}, ..., x_n)$, it is necessary to perform an integration which may be processed in parallel.

- **Configuration variable level:** in the case of some configuration variables $(q_1, ..., q_r)$, the convolution must be performed; trivial parallelization exists for configuration variables where there is no convolution $(q_{r+1}, ..., q_m)$.

The three levels previously stated are valid for any robotic structure, and different solutions may take advantage of one, two or three levels of parallelism at a time, resulting in faster C-space evaluation algorithms.

3 Case Study: 3D Mobile Robot

A good example of application of the previous formalism is the evaluation of the configuration space for a mobile robot, which moves on a 3D workspace partially occupied by obstacles. In [8] [12] an algorithm is proposed for calculating a bitmap that represents the C-space of a mobile robot as the convolution of two bitmaps (one representing the obstacles in the workspace and one representing the robot).

The expression for evaluating the C-space for this robot can be followed from (4) and (5). Let be (x_r, y_r, θ_r) the parameters that define a configuration q, being (x_r, y_r) the coordinates of the center of the robot and θ_r its orientation. The function $CB(x_r, y_r, \theta_r)$ would be given by

$$\int A(x_r, y_r, \theta_r, x, y, z) B(x, y, z) dxdydz$$

The convolution product of two functions defined in R^3 is obtained, but only over x and y; it is necessary to make the integral of this product over z. So,

$$CB(x_r, y_r, \theta_r) = \int (A_{(0,0,\theta_r)} * B)_{(x,y)}(x_r, y_r, z) dz$$

For each orientation θ_r of the robot, a slice $CB(x_r, y_r, \theta_r)$ is obtained. Each one is calculated by adding the convolution products of $A_{(0,0,\theta_r)}$ and B for each plane $z = constant$, from $z = 0$ to $z = robot\ height$. Applying the convolution theorem the C-obstacles are obtained by the inverse Fourier transform of

$$\int \mathcal{F}\left[A_{(0,0,\theta_r)}(x_r, y_r, z)\right] \mathcal{F}\left[B(x_r, y_r, z)\right] dz$$

The algorithm proposed in [5] for the 3D mobile robot is presented in figure 1, where W is the bitmap of the workspace; $A_{(0,0,\theta_r)}$ represents the robot bitmap for a specific orientation θ_r; and the C-space is obtained in the calculated bitmap, CB.

As it can be observed, an iterative calculation of the C-space slice pertaining to each robot orientation θ_r is needed. The number of necessary iterations depends on the chosen discretization (N) to cover the interval robot orientations. For the spatial variable (z), where there is no convolution, an accumulation of convolution products is performed (Only the products up to the highest point of the robot are needed).

```
For each coordinate $z$
    Construct $W(x_r, y_r, z)$, placing '1' at its limits
    Compute two-dimensional $\mathcal{F}[W(x_r, y_r, z)]$
For every values of orientation $\theta_r$
    $PT = 0$
    For each coordinate $z$
        Construct $A_{(0,0,\theta_r)}(x_r, y_r, z)$
        Compute two-dimensional $\mathcal{F}\left[A_{(0,0,\theta_r)}\right]$
        Let $P = \mathcal{F}[W] \cdot \mathcal{F}\left[A_{(0,0,\theta_r)}\right]$
        Accumulate at $PT$
    Let $IP(x_r, y_r) = \mathcal{F}^{-1}[PT]$ (the inverse FFT)
    Let $CB(x_r, y_r, \theta_r) = 1$ iff $|IP(x_r, y_r)| > 0$
```

Figure 1: Sequential algorithm

4 Parallel Algorithm Implementation

In this case the three levels of paralelism stated are present: *configuration variable level*, the algorithm performs a series of independent operations, one for each robot orientation; *spatial variable level*, each orientation needs to accumulate the convolution products for each plane z, that can be done in parallel; *Fourier transform level*, which is always present.

Several designs that take advantage of the inherent paralellism has been proposed in [13]; however, due to the available test environment, a tetra-processor machine, we used a simplified proposal: the work load is big enough as to be more efficient to exploit only one level (Figure 2); otherwise, the performance would suffer from the cost and complexity of communications.

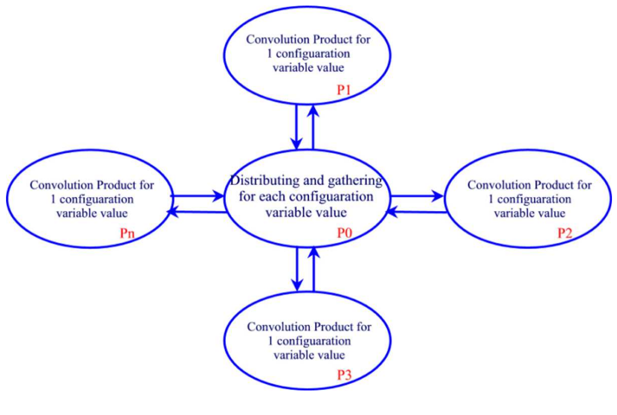
Convolution Product for 1 configuaration variable value
P1
Convolution Product for 1 configuaration variable value
Pn
Distributing and gathering for each configuaration variable value
P0
Convolution Product for 1 configuaration variable value
P2
Convolution Product for 1 configuaration variable value
P3

Figure 2: Parallel design at configuration variable level

According to [6], the master-slave paradigm is a suitable choice, since a master process is going to distribute tasks to each of the slave processes and, gradually, receive the slaves calculations for building the final result (Figure 2). In this case we understand by *task* the necessary calculations to obtain a slice of the C-space (Figure 2), and by *grain size* the number of orientations that constitute a task. This means an adequate load balance if as many slaves as the available number of processors are utilized, taking into account that the calculation time of the tasks is over the employed time in the exchange of information. Previous works ([13]) taught that the best performance is achieved when working with fine grain, i. e., just one orientation.

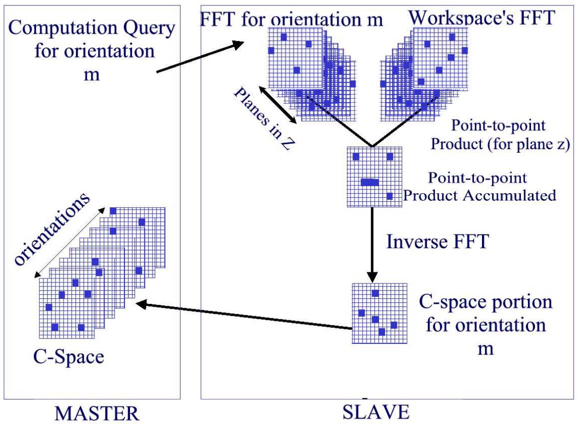
Computation Query for orientation m
FFT for orientation m
Workspace's FFT
Planes in Z
Point-to-point Product (for plane z)
Point-to-point Product Accumulated
Inverse FFT
orientations
C-space portion for orientation m
C-Space
MASTER
SLAVE

Figure 3: Basic scheme for the *C-space* parallel calculation

In figure 2 the essential operations that carry out the master and the slaves processes are shown. The master takes charge of calculating the preliminary data that slaves need, it directs the execution, it marks the finish of all the processes, and it builds the final result. On the other hand, each slave calculates the C-space slice pertaining to the orientation assigned by the master and provides the result.

4.1 Master Process

In the algorithm of figure 1 it is observed that in each iteration, the Fourier transform of the workspace bitmap is utilized, that is, it should be available for each slave before any calculation. Therefore, the master should obtain the bitmap of the workspace and evaluate its Fourier transform. For that, it needs to calculate the work table for the execution of the FFT. It must be stressed that the slaves will also need this table for the execution of FFTs. All this information, calculated only once, is sent to each of the slaves.

From this moment the master is going to communicate to each slave the task that has to be calculated, receive the result (the slice of the corresponding *C-space*) and assign a new task. Finally, it asks them to finish when there is no more orientations to calculate. During this process the master is going to build a tridimensional array representing the C-space.

4.2 Slave Process

In order for the slave to calculate each portion of the C-space, it must know which orientation is involved (determined by the master with the number of task). Thus, the slave builds the needed robot bitmap for that orientation and a z plane and obtains its FFT (Figure 3). Later, it carries out the point-to-point product of this FFT with the transformed workspace that the master has provided. This procedure is repeated a number of times as determined by the highest point of the mobile robot, and the results are added (i.e., the accumulation of the spatial variable level, which in this case, as stated before, it is not exploited). Next, it applies the inverse transformation to the accumulated result to obtain the portion of the C-space. This intermediate result is what the slaves, in turn, send to the master, expecting another task to be assigned or an end of execution signal.

5 Results

A Silicon Graphics Origin 200 computer with four MIPS R10000 processors and 768 Mbytes of memory has been used to validate the implementation. We have used the MPI (*Message Passing Interface*) tool to implement the parallel algorithm.

In figure 4 a complex scenario is shown with several polyhedral obstacles and the studied 3D mobile robot, using bitmaps with a resolution of 128x128x128. The corresponding C-space is shown in Figure 5 where a small set of free collision configurations appears inside a big C-obstacle; the holes in the big C-obstacle correspond to the configurations of the robot where it is positioned between the two *column-like* obstacles or between a *column-like* obstcale and the *bridge-like* one.

The best results have been obtained for the case of one master process and four slaves processes when working with fine grain, i.e., just one orientation (reducing the average computation time to less than a third part).

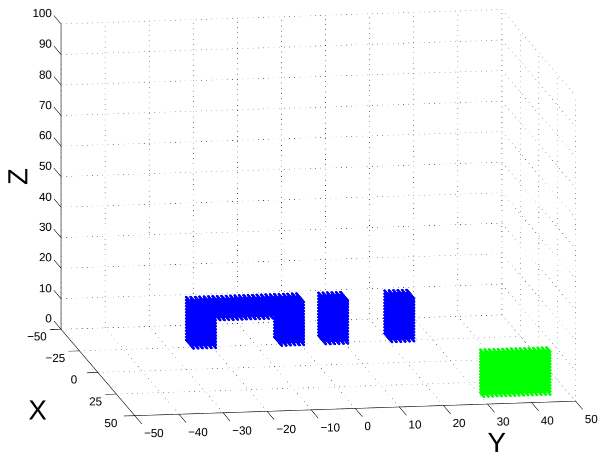

Figure 4: A set of obstacles and the 3D mobile robot

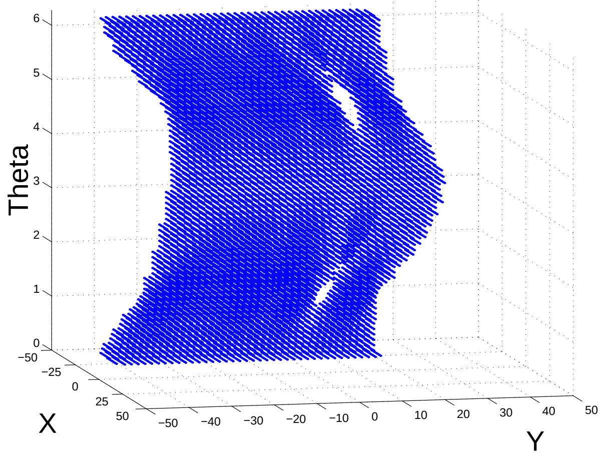

Figure 5: C-obstacles for the 3D mobile robot

In Tables 1, 2 and 3, the obtained calculation times (in seconds) are shown for the execution of the fine grain parallel algorithm and the sequential algorithm for different resolutions and distinct heights (number of pixels in the bipmap) of the robot.

Sequential	Parallel	Robot height	Speedup
0.99	0.73	5	1.36
3.34	1.23	20	2.72
7.30	2.40	45	3.04
10.51	3.21	64	**3.27**

Table 1: Execution times (s) for 64 x 64 resolution

It can be observed a growing tendency in the speedup when increasing the resolution from 64x64 to 128x128; most likely the trend comes from communications penalties when the time dedicated to pure calculation is small, in the case of a resolution of 64x64 in the bitmaps. However, Table 3 shows a smaller performance due to memory availability. On the other hand, when the robot is higher, that is, there are more slices to calculate, the speedup is also better.

In Figure 6 it can be seen the parallel execution of the algorithm for five processes running on four processors: one master and four slaves. A color code is used: waiting periods are represented in dark grey; working periods are shown in light grey; and white is used for the time dedicated for MPI functions. The first bar (number 0) is the master and the others (1 to 4) are the slaves. First, it can be observed in all processes an initial white period followed by a dark grey period; this is the required time for the intialitations and for all the slaves to receive its first task. After that, the master is nearly all the execution dark grey; after sending a new orientation to a slave, it must wait for another slave to finish, receive the slice, and assign new task. On the other hand, slaves are most of the execution light grey, which means that they are always working, stoping only for comunicating the partial results.

As Figure 6 and resulting speedups reflect, the performance is optimal: slave processes are working most of the time and the master process is waiting for the

Sequential	Parallel	Robot height	Speedup
14.30	5.22	10	2.74
52.81	15.21	40	3.47
85.05	23.45	64	3.63
164.06	44.34	128	**3.70**

Table 2: Execution times (s) for 128 x 128 resolution

Sequential	Parallel	Robot height	Speedup
252.45	76.67	15	3.29
376.54	107.51	30	3.50
792.70	226.77	64	3.50

Table 3: Execution times (s) for 256 x 256 resolution

finalization of the assigned tasks.

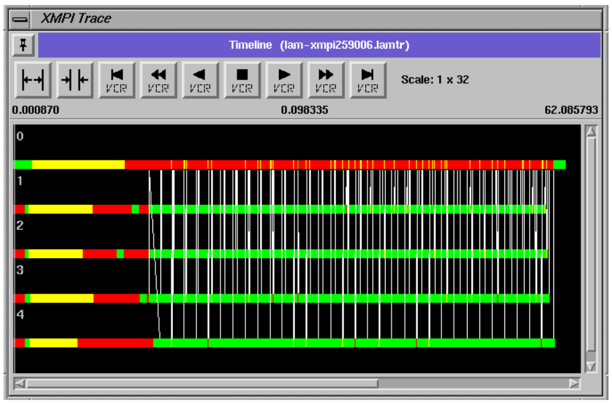

Figure 6: MPI execution: 1 master and 4 slaves

6 Conclusions

A design and implementation of the algorithms to evaluate the robot C-space in parallel has been presented. The algorithms are based on the evaluation of the equations expressed within a general mathematical formalism. As a practical application, a three dimensional mobile robot is considered. Limitations of test environment prompted development and implementation of a solution optimizing at the configuration variable level. Different experiments have been carried out in which the resulting speedups are very acceptable.

The presented case study provides reasonable results that could be extrapolated to more complex robotic structures, so they can hopefully be designed on the lines of the parallel algorithm studied in this paper.

Acknowledgements

We want to acknowledge the partial support given by the JCyL through SA02-00F research project.

References

[1] Valmir C. Barbosa. *An Introduction to Distributed Algorithms*. The MIT Press, 1991.

[2] G. Bilardi, S. Hornick, and M. Sarrafradeh. Optimal VLSI architechtures for multidimensional DFT . *ACM Comp. Architecture News*, 19(1):45–52, 1991.

[3] D. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proceedings of IEEE Int. Conference on Robotics and Automation*, volume 2, pages 46–51, 1993.

[4] D. E. Culler and J. P. Singh. *Parallel Computer Architecture.* Morgan Kaufmann, S F, 1999.

[5] B. Curto, V. Moreno, and F. J. Blanco. A general method for c-space evaluation and its application to articulated robots. *IEEE Transactions on Robotics and Automation*, 18(1):24–31, 2002.

[6] I. Foster. *Designing and Building Parallel Programs. Concepts and Tools for Parallel Software Engineering.* Addison-Wesley P.C., 1995.

[7] D. Henrich. Fast motion planning by parallel processing. a review. *Journal of Intelligent and Robotic Systems*, 20(1):45–69, 1997.

[8] Lydia E. Kavraki. Computation of configuration space obstacles using the fast fourier transform. *IEEE Tr. on Robotics and Automation*, 11(3):408–413, 1995.

[9] J. C. Latombe. *Robot motion planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[10] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32:108–120, 2 1983.

[11] T. Lozano-Pérez and P.O'Donnell. Parallel robot motion planning. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1000–1007, 1991.

[12] R. Thern, Francisco Javier Blanco, Beln Curto, and Vidal Moreno. Evaluation of the configuration space of robots upon a distributed computing system. In *Proceedings of the IEEE-IFAC European Control Conference*, Porto, Portugal, september 2001.

[13] R. Theron, F. J. Blanco, B. Curto, V. Moreno, and F. J. García. Parallelism and robotics: The perfect marriage. *ACM Crossroads*, 8.3 Parallel computing, 2002.