

# Parallel Motion Planning with the Ariadne's Clew Algorithm

E. Mazer, J.M. Ahuactzin, E. Talbi, P. Bessiere and T. Chatroux

Laboratoire d'Informatique Fondamentale et d'Intelligence  
Artificielle  
46 Avenue Felix Viallet  
38000 Grenoble, France

**Abstract**— We describe an implementation of a real time path planner for a robot arm with six degrees of freedom moving among dynamical obstacles. The planner is based on a novel technique called the Ariadne's Clew Algorithm. A brief description of this algorithm and parallel implementation of it are presented. Finally we analyze experiments made with this planner.

## 1. Introduction

Our experimental testbed includes two robot's arms each having six degrees of freedom. Our goal was to plan the motions of one of these robots (the Controlled Robot) with a path planner fast enough to "re-plan" in line new trajectories when the other arm (the Dynamical Obstacle Robot) was placed on its way towards the goal. The idea was to demonstrate the possibility of including a global path planner into an incremental trajectory generator. This idea was first successfully experimented with a 2D version of our planner. A simulated mobile robot uses this planner to plan its trajectories among obstacles. Using the simulator the user was able to change the positions of the obstacles while the mobile robot was moving towards the goal, in that case, the planner reacts immediately by re-computing a new trajectory from its current position to the goal. For example if the door represented figure 1 was closed before the robot reached it, the planner was quickly (in less than 0.05s) able to devise a new path. As a result the robot was continuously progressing towards the goal despite the tentatives of the operator to move obstacles on its way. This report describes an attempt to reach the same level of reactivity with a six degree of freedom arm moving in a three dimensional environment.

## 2. Previous Work

A recent survey of robot motion planning techniques can be found in [2]. A detailed presentation of the field exists also in [3]. Many attempts have been made to obtain "real-time" path planners. For example, in the proceedings of the last conference on Robotic and Automation one can find: a method to implement the planner of Latombe and Baranquant on a parallel machine [4], a method which uses an optically computed potential fields [5] and three "fast

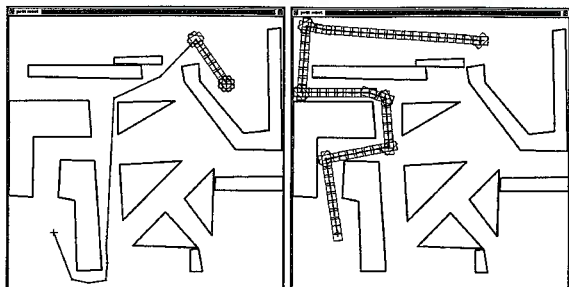


Figure 1. Re - planning when a door is closed

path planners" [7, 8, 6]. However, while we try to reach the same goal, the work described in this paper is more related to the approach taken by Overmars in [9] and to the Sandros [10] motion planner. Both methods use a set of landmarks to represent the free space and a local planner which is used to connect the landmarks as well as the initial and final positions. In the work described by Overmars, the landmarks are placed randomly in the search space, when a landmark is placed into a C-Obstacle it is moved to a close free location. In the Sandros motion planner the landmarks are placed in "slices" of the configuration space. The completeness of both methods have been proved, and fast response time have been reported. The main advantage of our planner is its ability to place landmarks more efficiently than the previous methods and, thanks to an implementation on a massively parallel machine, it is also much faster.

### 3. The Ariadne's clew algorithm

A detailed description of the Ariadne's clew algorithm can be found in [11, 13], here we only sketch its principle. The ultimate goal of a path planner is to find a path in the configuration space from the initial position to the target. However, while searching for this path, an interesting sub-goal may be to try to collect information about the free space and about the possible paths to go about that space. The ARIADNE'S CLEW algorithm tries to do both at the same time and it is made of two sub-algorithms: SEARCH and EXPLORE. The EXPLORE algorithm collects information about the free space with an increasingly fine resolution, while, in parallel, the SEARCH algorithm opportunistically checks if the target can be reached. The EXPLORE algorithm works by placing landmarks in the search space in such a way that a path from the initial position to any landmark is known. In order to learn as much as possible about the free space the EXPLORE algorithm tries to spread the landmarks all over the space. To do so, it tries to put the landmarks as far as possible from one another. For each new landmark produced by the EXPLORE algorithm the SEARCH algorithm checks with a local method if the target may be reached from that landmark. Both the EXPLORE and the SEARCH algorithms may be seen as solving optimization problems on a special

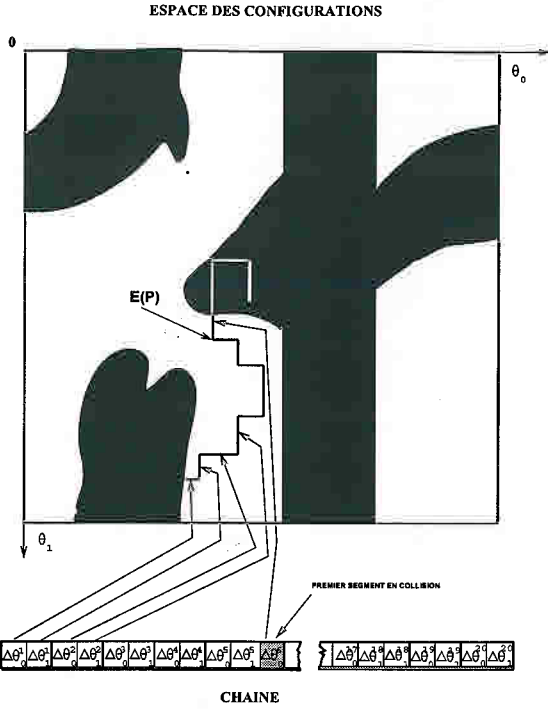


Figure 2. A Manhattan path in the configuration space

set: namely the set of **Manhattan paths** of fixed length.

### 3.1. The Manhattan paths

Let  $(\theta_1, \dots, \theta_n)$  denotes the configuration of a system with  $n$  degrees of freedom. We define a Manhattan path of length 1 as the path consisting of moving each degree of freedom once. We denote such a path as:

$$p^1 = \{\Delta\theta_1^1, \Delta\theta_2^1, \dots, \Delta\theta_i^1, \dots, \Delta\theta_n^1\}$$

More generally we define a Manhattan path of length  $k$  as the concatenation of  $k$  Manhattan paths of length 1.

$$p \in \mathfrak{R}^{k \times n} = \{\Delta\theta_1^1, \dots, \Delta\theta_i^j, \dots, \Delta\theta_n^k\}$$

Given a Manhattan path  $p$  of length  $k$  we denote by  $E(p)$  the point of the configuration space corresponding to the extremity of the last collision free segment of  $p$  (see figure 2). Basic geometric computations [1] permit to compute  $E(p)$ .

### 3.2. SEARCH

Given a goal configuration  $\Theta^g$  we define a function  $F : \mathfrak{R}^{k \times n} \xrightarrow{F} \mathfrak{R}^+$  as :

$$F(p) = \|E(p) - \Theta^g\|$$

If a Manhattan path  $p_0$  towards the goal exists then  $F(p_0) = 0$ . As a result, finding a Manhattan path  $p$  of length  $k$  can be seen as a minimization problem:

$$\min_{p \in \mathfrak{R}^{k \times n}} F(p)$$

While we do not have any analytical expression for  $F$  we can easily compute its value at any point of  $\mathfrak{R}^{k \times n}$ . Many methods exist to minimize such a function, we use a genetic algorithm. Even in the presence of local minima, we can implicitly define a region of  $\mathfrak{R}^{k \times n}$  called the "back-projection" of  $\Theta^g$  on which the minimization method find a global minimum for  $F$ . The goal of EXPLORE is to place a reachable point in that region.

### 3.3. EXPLORE

One can visualize the EXPLORE algorithm by imagining a robot placing landmarks in the free space starting from its initial position. Each time it places a new landmark it tries to place it as far as possible from landmarks previously placed. Each of the landmarks is connected by a free path to at least one of the other landmarks. We denote by  $EL_q$  the set of existing landmarks at step  $q$ . The explore-algorithm begins with  $EL_1 = \{L_1\}$  were  $L_1$  is the initial location then, in next step,  $EL$  is incremented with the new landmark. If the search space is bounded, then the robot will fill the free space connected to its initial position with landmarks. These landmarks will become closer and closer as the search time is becoming larger. At a given point in time the last generated landmark will necessarily fall into the back-projection region of the goal and a solution will be found !

We denote by  $PL$  the set of all Manhattan paths starting from the landmarks  $L_i \in EL_q$ ,  $PL$  is indexed by  $[0, 1, \dots, q] \times \mathfrak{R}^{k \times n}$ . We denote  $E(p)$  the extremity of the last non-colliding segment of a particular path  $p \in PL$ .

Let  $p_2$  the path of  $PL$  which maximize

$$p_2 : \max_{p \in PL} \|L_1 - E(p)\|$$

According to our definitions, the point  $L_2 = E(p_2)$  is the most further location of the search space reachable from the initial location  $L_1$ , we choose it as the second landmark. Now, we have have  $EL_2 = \{L_1, L_2\}$ . Given a path  $p$  starting either from  $L_1$  or  $L_2$  we consider the minimum value between  $\|E(p) - L_1\|$  and  $\|E(p) - L_2\|$  and we try to maximize this value over  $EL_2$  in order to find a new reachable landmark which is as far as possible from  $L_1$  and from  $L_2$ . In other word:

$$p_3 : \max_{p \in PL} \min\{\|L_2 - E(p)\|, \|L_1 - E(p)\|\}$$

More generally if we have already  $n$  landmarks we can find the  $n + 1$  landmark by maximizing the following expression:

$$p_{n+1} : \max_{p \in PL} \min_{j=1, n} \| L_j - E(p) \| \quad (1)$$

By taking  $L_{n+1} = E(p_{n+1})$  we get our new  $n + 1^{th}$  landmark. Lets consider the function:

$$\forall n \geq 2 : V(n) = \max_{p \in PL} \min_{j=1, n-1} \| L_j - E(p) \|$$

If the search space is bounded then:

$$\lim_{n \rightarrow \infty} V(n) = 0$$

or

$$\forall \varepsilon \exists n : \forall j > n \quad V(j) < \varepsilon$$

Then, if  $G$  is a point of the accessible free space (ie: it exists a path from  $L_1$  to  $G$ ) then we have:

$$\forall \varepsilon : \exists n \| L_n - G \| < \varepsilon \quad (2)$$

We call this property : *epsilon-reachability*. The epsilon-reachability has a strong consequence for planning a path in a continuous space : if one can find  $\varepsilon$  such that there is a function which solve the path planning problem in any ball of diameter  $\varepsilon$  (the search function), then by combining it with the explore algorithm we get a deterministic method to plan a path between any two points of the configuration space.

### 3.4. A Parallel Implementation

We have implemented this algorithm on a parallel machine with 128 transputers. The figure 3 represents the configuration we choose to implement the Ariadne's clew algorithm. The numbers inside the rectangles indicate the physical number of each processor, the edges correspond to the physical links between two transputers. It is possible to use a programmable switch board to configure the machine with this particular topology. In this architecture one has to consider three levels of parallelism.

1. **The parallel execution of "Search" and "Explore".** At the first level the EXPLORE algorithm, which uses the processor 1 to 60, runs in parallel with the SEARCH algorithm (processors 61 to 120). The processor 128 is used by EXPLORE to communicate the landmarks to SEARCH.
2. **The parallel execution of the genetic algorithm.** Both EXPLORE and SEARCH uses a genetic algorithm as an optimization technique. A description of parallel genetic algorithms can be found in [12]. In our case the population used by the genetic algorithm is reduced to six

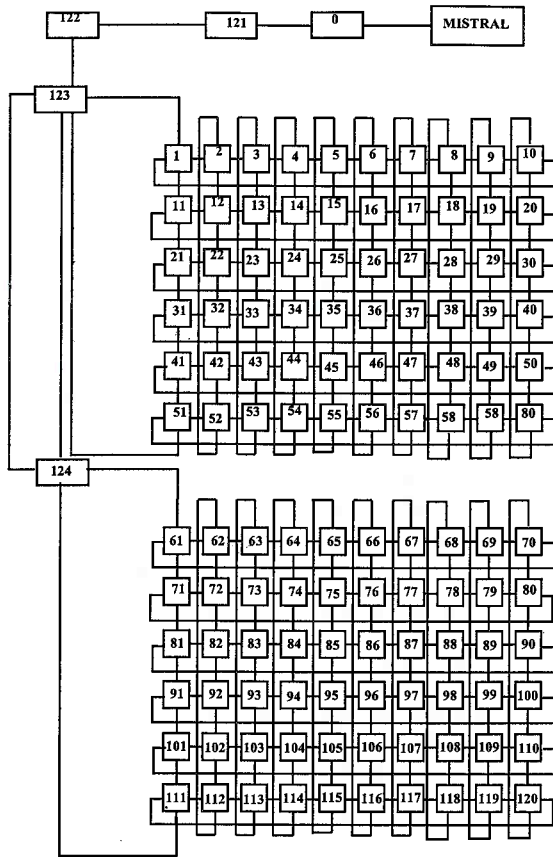


Figure 3. Actual implementation on 128 Transputers

individual. Each individual evaluates in parallel the cost function with its own "farm" of processors. For examples the "individuals" of SEARCH are located on processors 1-11-21-31-41-51 and are organized in a single ring (see figure 4).

3. **The parallel evaluation of the cost function.** In both cases, the evaluation of the cost function boiled down to many serial computations of the legal range of motion for a single link. In turn, the computation of each legal range can be split into three types of elementary range computation denote as A,B and C. For each individual the computational load is spread over its farm of processors by having each processor of the farm responsible for a given elementary computation. For example the processor 21 uses the processors 22-23-24 to perform the computation of type A, the processors 25-26-27 for the computation of type B and the

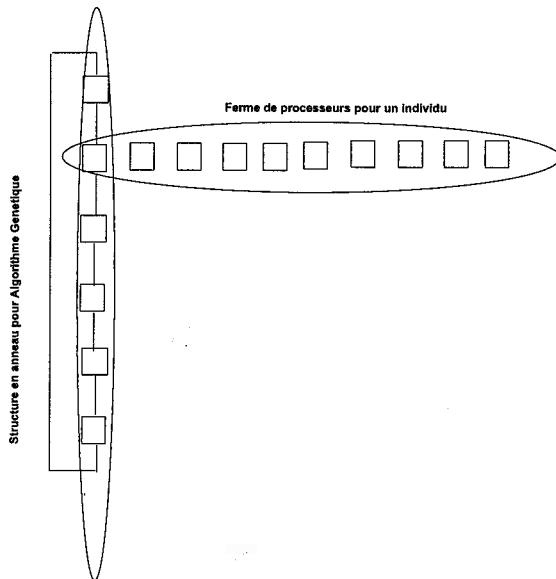


Figure 4. Ring structure and processor farm

processors 28-29-30 for the type C. .

#### 4. Experiments

The figure 5 represents the architecture of our experiment setup.

The robot I is under the control of the Mega-Node (via Kali) running a parallel implementation of the Ariadne's Clew algorithm. The robot II is used as a dynamical obstacle: it is controlled (via Kali) by a random motion generator.

First we use our robot simulation package ACT to describe the scene with the two robots. We compile this representation into a special geometrical representation which enclosed each obstacle and each link of the robots into a box to minimize the number of geometric computation.

This model is downloaded to all the processors as well as the goal position for robot I, then the following algorithm is executed :

1. **Step 1** : The current position of the two robots is diffused to all the processors.
2. **Step 2** : The Mega-node produces a plan which assumes robot II is standing still.
3. **Step 3** : Kali executes only the first part of the plan (a manhattan pass of length one).

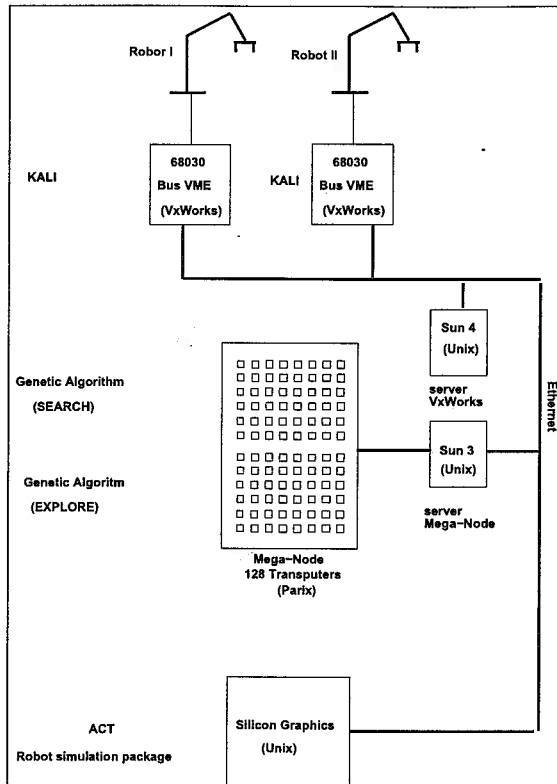


Figure 5. A parallel Architecture for Robot Motion Planning

4. **Step 4** : The random motion planner is called and produces a random motion which is executed by robot II (with robot I standing still).
5. **Go to Step 1**

## 5. Experimental Results

To speed up the computation of the evaluation function we only use the enclosing boxes of the obstacles and of the links. The controlled robot is made of 6 boxes and there are 10 boxes used as obstacles including the moving obstacles (see figure 6).

In that particular environment, a Manhattan path of length one requires  $\frac{6^2+6}{2} * 10 = 210$  computations for the rotational ranges. Each computation of the range requires 240 elementary contact computations. So, to evaluate a path of length 5, 252,000 elementary contact computations are necessary. To perform the evaluation of a single generation of the genetic algorithm with 6 individuals we reach a total 1,512,000 contact computations. It would not be possible

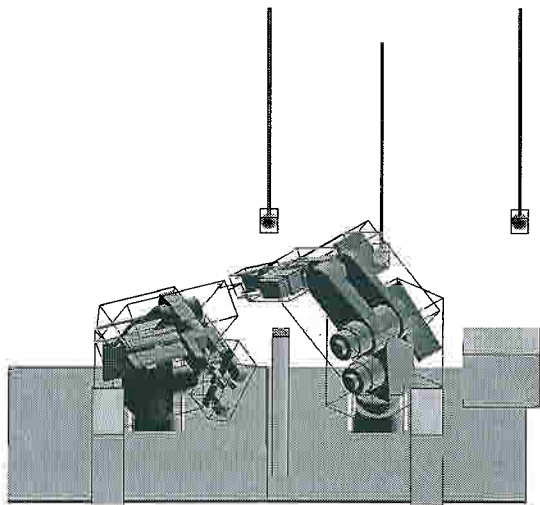


Figure 6. The experimental scene with enclosing boxes

to handle such a large amount of computation without the use of geometric filters which permit to discard quickly the link-obstacles pairs which cannot interact. In practice the use of geometric filters permit to reduce the amount of "real" geometric computation by a factor 10. However the computational load remains very heavy, even for a simple scene such as the one we use in our example. Only the use of a massively parallel machine permits to keep the planning time reasonably small : between 0.5 to 3 seconds for a six degrees of freedom arm with 10 obstacles. It is also important to note that the number of generations used by the genetic algorithm to produce a solution is very small : 5. As a consequence we believe that very little optimization is made with the genetic algorithm, it seems that the genetic algorithm is only useful to find the most promising landmark which is "closer" from the free space not already explored. Nevertheless we have successfully tried the Ariadne's clew algorithm on very complex scene necessitating non-trivial motion to reach the goal. This suggest that it may be possible to randomly find a path in a scene considered as "difficult" (note that placing landmarks is still a very useful process). We conjecture that many planners found in the literature are tested on scene where many randomly generated paths will lead to a point in the back-projection of the goal.

For an industrial application of path planning the number of potentially colliding (link-obstacle) pairs is several order of magnitude bigger than the one used in our experiment (210). In industry the models of the robot and the models of the obstacles are very detailed and contains hundreds of geometrical entities. In that case, the evaluation of a single Manhattan path will be impossible even on massively parallel machine.

## 6. Conclusion

We have presented a new motion planner for a six degrees of freedom arm. The planner achieves fast response time compare to existing planner but remains slow and cannot be used as part of a trajectory generator. Given a fixed number of obstacles the planning time is proportional to the difficulty of the task. The planning time is also proportional to the number of link-obstacles pairs found in the scene. We are currently working on a new version of the ARIADNE'S CLEW algorithm which will use a classical optimization algorithm as well as a hierarchy of geometrical models for the environment.

## References

- [1] T. Lozano-Pérez : *A simple motion-planning algorithm for general robot manipulators.*, IEEE Int. Jour. on Robotics and Automation, RA-3, June 1987.
- [2] Yong K. Hwang and Narendra Ahuja : *Gross Motion Planning - A Survey.*, ACM computing Surveys, Vol 24, No 3, pages 119-289, September 92.
- [3] Jean-Claude Latombe: *Robot Motion Planning*, Ed. Kluwer Academic Publisher, 1991.
- [4] D.J Challou, M. Gini, and V. Kumar : *Parallel Search Algorithms for robot Motion Planning* IEEE Int. Conf. On Robotics an Automation, Atlanta, May 93.
- [5] M.B Reid : *Path planning Using Optically Computed Potential Fields* IEEE Int. Conf. On Robotics an Automation, Atlanta, May 93.
- [6] C.W Warren *Fast path Planning Using Modified A\* Method* IEEE Int. Conf. On Robotics an Automation, Atlanta, May 93.
- [7] P.K Pal and K. Jayarajan : *Fast Path planning for Robot Manipulators Using Spatial Relations in the Configuration Space* IEEE Int. Conf. On Robotics an Automation, Atlanta, May 93.
- [8] P.Fiorini and Z.Shiller : *Motion Planning in Dynamic Environments Using the Relative Velocity Paradigm* IEEE Int. Conf. On Robotics an Automation, Atlanta, May 93.
- [9] M.H. Overmars : *A Random approach to motion planning* Spring School on Robot motion planning, Rodez(France), March 93.
- [10] P.C Chen and Y.K Hwang : *SANDROS: A motion planner with performance proportional to task difficulty* IEEE Int. Conf. On Robotics an Automation, Nice, May 92.
- [11] J. Ahuactzin, G. Talbi, P. Bessière and E.Mazer : *"Using Genetic Algorithm for robot motion planning"*. ECAI, 92 Vienne 1992.
- [12] G. Talbi, J. Ahuactzin, P. Bessière and E.Mazer : *"A parallel implementation of a robot motion planner"*. CONPAR92, Lyon, 1992.
- [13] E.Mazer, G. Talbi, J. Ahuactzin, P. Bessière : *"The Ariadne's Clew Algorithm"*, SAB92 Honolulu 1992.