LaTeX documents for the Machine Intelligence
and Pattern Recognition series
W.j. Maas

Toward Real-Time Motion Planning

Daniel J. Challou, Maria Gini, and Vipin Kumar *

Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, USA

Abstract

We show that parallel search techniques derived from their sequential counterparts can enable the solution of motion planning problems that are computationally impractical on sequential machines. We present a parallel version of a robot motion planning algorithm based on "quasi best first" search with randomized escape from local minima and random backtracking, and discuss its performance on a variety of problems and architectures.

1. Introduction

Among the many skills autonomous entities require to support their activities is the ability to plan the paths they must take while conducting those activities. Motion planning enables an object to move safely through its environment while achieving specific goals.

Motion planning algorithms are of great theoretical interest, but are rarely used in practice because of their computational complexity. In this paper we show how the use of multicomputers and appropriate parallel algorithms can substantially reduce the computation time required to synthesize paths for robots with a large number of joints. This speedup in computation enables the solution of motion planning problems that are, for all practical purposes, computationally impractical on sequential computers. Easier problems can be solved in a matter of seconds or less, thus real-time motion planning is an achievable goal.

*This work was supported in part by a contract between the Army Research Office and the University of Minnesota for the Army High Performance Computing Research Center. Additional support was furnished by NSF/CDA-9022509, IST/SDIO grant No. 28408-MA-SDI, and the Center for Advanced Manufacturing, Design and Control of the University of Minnesota.

We have implemented a parallel version of a motion planning algorithm based on "quasi best first" search with randomized escape from local minima and random backtracking. We have analyzed its performance on a variety of problems and architectures including the nCUBE2[1] multicomputer (with up to 1024 processors), the CM-5[2], and a network of workstations. The method delivers excellent speedup and appears very promising [5].

1.1. Related work

Research in the area of robot motion planning can be traced back to the late sixties, but most of the work has been carried out more recently. Over the last few years the theoretical and practical understanding of the issues has increased rapidly, and a variety of solutions have been proposed. Latombe [11] provides an extensive overview.

Motion planning algorithms usually formulate solutions to a given problem in a robot's configuration space (C-space). We present a brief description of C-space here in order to clarify the discussion that follows. The *workspace* of a robot is the world that it is capable of moving through; the workspace usually includes objects or obstacles. A *configuration* of a robot is the specification of the position and orientation of the robot with respect to a fixed reference frame. The *configuration space* (C-Space) of a robot is the set of all configurations that can be assigned to it.

The reason we concern ourselves with the space of possible configurations is that in such a space the robot becomes a point. The *free C-Space* is the set of configurations belonging to the robot's C-Space in which the robot is not in a state of collision with itself or any other object in its workspace. The dimensionality of the C-Space is the number of parameters required to fully specify a configuration of the robot. For example, a fixed base robot arm with six degrees of freedom (i.e. six joints), such as the one pictured in fig. 2, operates in a six dimensional C-Space.

Many motion planning algorithms decompose the search space into discrete components called cells [11]. The motion planning problem then becomes one of computing a decomposition and searching through sequences of contiguous cells to find a path through free space (i.e. a sequence of configurations that involves no collisions with obstacles). Unfortunately, as more degrees of freedom are added to the object most methods become computationally impractical [16].

---

[1] nCUBE2 is a registered trademark of the nCUBE corporation

[2] CM-5 is a registered trademark of the Thinking Machines Corporation. The results obtained on the CM-5 that are presented in this paper are based upon a beta version of the software and, consequently, are not necessarily representative of the performance of the full version of the software.

Recently, Lozano-Perez [12] developed a parallel algorithm which computes the discretized C-space for the first three links of a six degree of freedom manipulator. The path for the gripper portion of the manipulator is found by computing its free C-space in parallel at each arm configuration considered by the sequential search algorithm. Although this method works well, it is limited to relatively coarse C-space discretizations (it has a maximum discretization level of 64) because of the lack of memory available in which to store the precomputed C-space on the target architecture.

Other than the parallel scheme developed by Lozano-Perez described above, we are not aware of other existing parallel methods capable of solving instances of the motion planning problem involving higher dimensional C-space. Fortunately a great deal of work has been done in developing parallel search algorithms capable of solving similar problems [10, 9]. Many of the algorithms developed deliver linear speedup with increasing problem and processor size on various problems [2, 8]. It would seem that parallel motion planning methods which use such parallel search schemes should be able to deliver such performance as well. This is due to the following observation.

Amdahl's law states that if s is the serial fraction of an algorithm then, no matter how many processors are used, speedup is bounded by $1/s$ [1]. Thus, if an algorithm spends 98 percent of its time computing a certain function (say C-Space), and only that component can be computed in parallel, then the maximum speedup which can be obtained by the parallel algorithm is 50 because it must still spend 2 percent of its time in its serial component. Hence, parallelizing robot motion planning methods that rely almost entirely on a totally parallelizable search process may yield better speedup than parallelizing those approaches with even a small serial component.

Deterministic parallel search schemes deliver good speedup only when good heuristics are available to guide the search process. Unfortunately no perfect hill climbing heuristics are known for the general motion planning problem. Ertel [6] has shown that randomized parallel search can be extremely effective on theorem proving problems where no good heuristic is available. More specifically Ertel showed that when purely random search is applied to theorem proving applications, the lower bound for the parallel run-time is equal to the shortest possible run time which can be delivered by a single processor executing random search. Natarajan [15] has shown that when n processors perform n identical tasks in order to solve the same problem, such as n processors executing randomized search, the speedup which can be obtained is $1/p$ where p is the probability of finding a solution to the problem.

These last results imply that parallel random search may be effective for solving problems with highly unstructured search spaces. We discuss some

Step I: Compute the heuristics used to guide the search:

(i) Pick a "Control" point (or points) on the robot.
(ii) Pick the desired goal location in the workspace for the point(s) designated in step 1.
(iii) *BROADCAST THE WORKSPACE BITMAP AND DESIRED GOAL LOCATION OF THE CONTROL POINT(S) TO ALL PROCESSORS.*
(iv) For each "Control Point": starting from the desired goal location, label each point in the workspace that is not an obstacle with the L1 (city block distance) from the goal location. For example, in a 2-D workspace, the goal location gets the value 0, its four neighbors get the value 1, etc.

Step II: Search using the heuristics computed in Step I to evaluate which new configuration to expand. This is done by the algorithm below.

```
trials = 0
path = start configuration
repeat
  path trials = 0
  temp_path = end of path
  repeat
    quasi best first search until a local minimum is reached
    if TERMINATION MESSAGE RECEIVED then exit
    brownian motion to escape local minimum
    if TERMINATION MESSAGE RECEIVED then exit
    if (path trials > threshold) then
      randomly backtrack to a previous point in temp_path
  until (path trials > max_better_path_trials
    or temp_path with new minimum found)
  if new minimum found then
    append temp_path to path
until (solution found or trials > trial_limit)
if SOLUTION FOUND then
  BROADCAST TERMINATION MESSAGE TO ALL
    OTHER PROCESSORS
```

Fig. 1. An outline of the parallel motion planning algorithm. The capitalized statements highlight the additions we made to the sequential algorithm in order to enable the it to run on an MIMD multicomputer.

of the initial results delivered by our parallel implementation in the next section.

2. A Parallel Motion Planning Algorithm for MIMD multicomputers

In this section we outline our parallel implementation of the randomized method proposed in [3]. Barraquand and Latombe describe two algorithms. The first algorithm utilizes best first search and is resolution complete[3], but becomes computationally impractical when the dimension of the configuration space exceeds four. The second algorithm utilizes a randomized search that is probabilistically complete[4], and, in general, runs much faster than the complete approach. Both algorithms utilize discrete representations of the robot, the robot's workspace, and its C-Space. Space is represented with multiscale pyramids of bitmap arrays.

As discussed earlier, there are two different spaces associated with motion planning algorithms: the workspace and the C-space. The workspace is the world that the robot must move through, the C-space is the space in which the search for a collision-free path is performed. Discrete (e.g. bitmap) representations of the workspace are especially convenient when sensory data are used to construct them [14]. Even though we are not constructing the representation of the obstacles in the robot environment from sensors, we are interested in working with a representation that can easily be used in conjunction with a variety of range sensors.

Artificial numerical potential fields are used as the heuristic to guide the search [3]. Since the search for a path is performed in C-space, one would expect the heuristics to be computed in C-space. However the size of the C-space grid increases exponentially with the degrees of freedom of the robot. This makes it impossible to precompute and store the C-space in advance for more than two or three degrees of freedom or for fine discretization levels. Thus, the C-space is not stored, but is generated as the search progresses.

An artificial potential field map is computed in workspace for each control point. A control point is a point on the robot whose desired goal location is specified in the workspace. Each cell in the numerical potential field map corresponds to a cell in the workspace. The value placed in a particular cell in the numerical potential field map depends on the location of obstacles and that cell's distance from the goal location in the workspace. Larger

[3] An algorithm is *resolution complete* if it is guaranteed to find a solution whenever one exists at the level of resolution used to represent the problem

[4] An algorithm is *probabilistically complete* when the probability of finding a path when one exists converges to 1 as the search time increases without bound.

Fig. 2 Start and Goal Configurations for Six Degree of Freedom Robot operating in a 256 x 256 cell workspace. Each C-space axis (joint) has 256 possible discrete positions. The base of the robot is fixed in the bottom-center in each frame of the picture.

No. Proc.	1	2	4	8	16	32	64	128	256	512
nCUBE2	8959	5002	1247	1103	397	204	59	39	32	27
CM-5	3543	1208	315	334	232	39	32	29	19	14
NWS	3088	793	567	247	180	NA	NA	NA	NA	NA

Fig. 3 Average solution times in seconds to solve the problem shown in fig. 2.

cell values indicate positions further away from the desired goal position.

The workspace potential field maps of each of the control points are then combined to produce the heuristics used to guide the search through C-space. The idea is to move through successively smaller artificial potential field values until a goal position is reached. If the value of a configuration is better than the value associated with its parent configuration, the configuration is checked for collisions, and added to the path. Otherwise another sibling of the current configuration is investigated.

Figure 1 outlines the algorithm. The capitalized statements in the algorithm highlight the additions we made to the sequential algorithm in order to enable it to run on an multiple instruction multiple data (MIMD) multicomputer. Each processor runs the same basic program. The only interprocessor communication done is a broadcast of the workspace bitmap

and desired goal location(s) of the control point(s) in the workspace to all processors in **Step I**, and checks for a message indicating that another processor has found a solution in **Step II**.

The search step described in **Step II** of fig. 1 is "quasi best first" because instead of generating all possible successors, we generate the successors randomly, evaluate them heuristically using the artificial potential field, and pursue them only if they have a better heuristic value than their parent configuration. If enough successors are generated in each iteration of the quasi best first phase, then the method approximates best first search.

A local minimum occurs when the configurations that succeed the current configuration are no closer to the desired goal position than the current configuration. When a local minimum is reached, a brownian motion (random walk) is executed and then quasi best first search is resumed. The search and random walk steps are repeated until a solution is found in the quasi best first search phase, or the time limit in which the solution must be found is exceeded. The "quasi best first" search and random walks are the means by which the search-space is partitioned, as they insure that each different processor searches different parts of C-Space.

2.1. Discussion of Results

The table in fig. 3 documents results delivered by the parallel planner on the problem instance pictured in fig. 2. The table summarizes the data for ten runs of the Quasi Best First/ Random Planner on up to 512 processors on an nCUBE2, 512 processors of a CM-5, and a Network of 16 Sun Workstations. The table shows the average time to find a solution on each of the different hardware platforms for the problem instance involving the six degree of freedom fixed-base robot arm operating in a 256 x 256 cell workspace pictured in fig. 2. Each C-space axis has 256 possible discrete positions. Entries labelled NA indicate that timings are not yet available for that number of processors. All times are in seconds.

At first one might be surprised that such a straight forward parallel algorithm fares as well as it does, reducing the average computation time on the CM-5 from almost 1 hour on one processor to an average of 14 seconds on 512 processors, and from almost 1 hour to 3 minutes on a network of 16 workstations.

In this example, the average time taken to solve the problem decreases and levels off as we increase the number of processors because we hit a point where the number of processors required to insure that one processor will find a solution in the minimum amount of time possible for the algorithm is near optimal or optimal. This is because the probability that the random component of the algorithm will ensure that different processors

are exploring different parts of the search space decreases as we add more processors. When we reach that point, then adding more processors to the problem will just result in more processors doing redundant work (in the average case).

This approach has delivered similar results on more difficult problem instances as well. On one particularly difficult problem instance discussed in [5], this approach reduced the solution time from an average of over 14 hours on one processor to 3 minutes on 1024 processors, and delivered superlinear speedup on on up to 256 nCUBE2 processors. On problems where coarser levels of C-Space discretization are sufficient, such as the opposite of the problem pictured in fig. 2, we have been able to obtain solutions in an average time of 6.9 seconds using 256 CM-5 processors. Considering that this last result was obtained with a C-Space discretization level of 128, and 128 is much finer than any other approach that we know of, we believe it will be possible to generate motion plans in the sub-second time frame by using even coarser levels of discretization and greater numbers of processors.

Another important property of this approach is that when it is executed with an increased number of processors, it tends to produce better solutions. Figure 4 shows that as the number of processors performing random search increases, the average solution path length constructed by the processor finding a solution first tends to decrease. We have observed this behavior in all the experiments we have performed to date. Moreover, the variance in time to solution behaves similarly, that is, it decreases as the number of processors attempting to solve the problem increases.

Technological advances have already increased the potential for this approach to formulate motion plans in real–time. Figures 3 and 5 show that the CM-5 needs approximately one half the number of processors to deliver about the same results as delivered by the nCUBE2. As more powerful microprocessors used to construct massively parallel machines become available, (such as Digital Equipment Corporation's Alpha Chip, which is more than twice as fast as the Sparc processors on the workstations we used), we expect that the time required to solve problems using this approach will continue to decrease accordingly.

One might argue that massively parallel machines are not a viable platform for motion planning systems due to their prohibitive cost (and limited availability). However, due to the continuing progress in VLSI design and economy of scale resulting from their widespread use, the cost of processors that massively parallel machines employ (such as the Sparc chip used by the CM-5) is expected to decrease. When this occurs, it will be feasible to build large scale parallel computers with substantial raw computing performance at a relatively small cost. Hence it is not at all unreasonable to

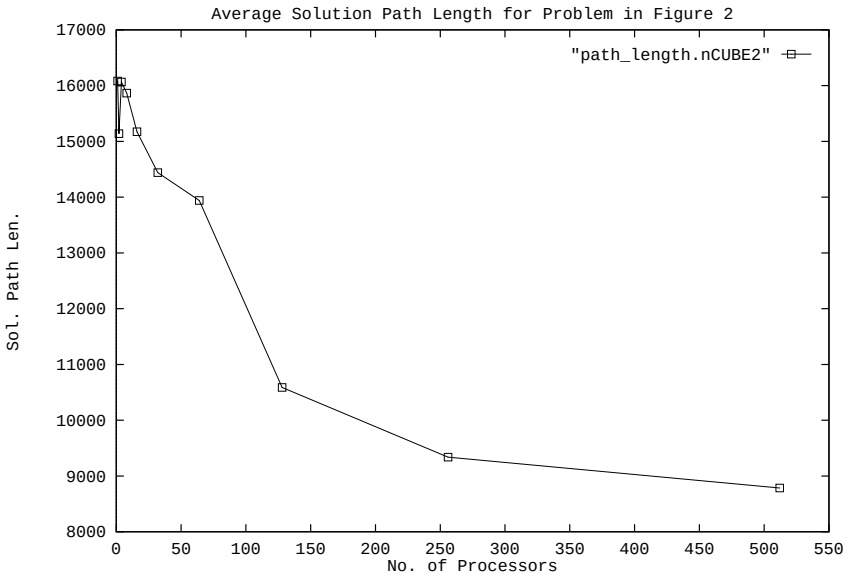

Fig. 4. Average solution path length found per number of processors used to solve the problem pictured in fig. 2 for the nCUBE2.

believe that the massively parallel machines we propose to use as the basis of our system will be readily available in the next decade.

Moreover, as the results on a network of workstations illustrate, one does not necessarily need a massively parallel machine to make significant reductions in the time it takes to solve fairly difficult problems. Assuming that a network of workstations continued to deliver results similar to those delivered by CM-5, then only 32 workstations would be required to reduce the computation time by two orders of magnitude. Currently, many companies, universities, and research institutions have such resources available.

3. Future work

As figure 6 shows, we are currently experimenting with a 3D version of the parallel planner on the CM-5. However, the randomized scheme we have implemented does have its drawbacks.

In many cases the paths the planner found were clearly sub-optimal in terms of length and their ability to be executed by any real robot. As

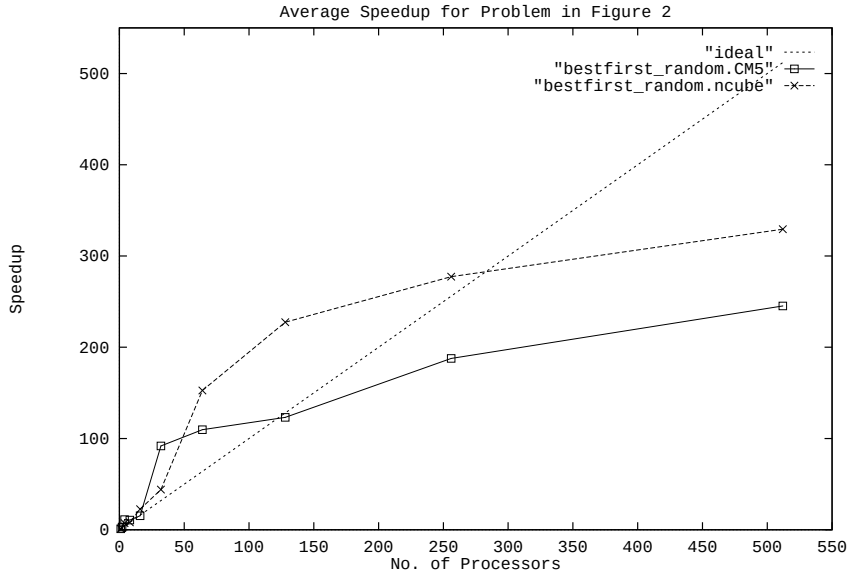

Fig. 5. Speedup on the nCUBE2 and CM-5 for the problem instance pictured in fig. 2.

Latombe et. al. have pointed out, in a substantial subset of such cases post processing can help optimize such paths [3]. Our preliminary results indicate that even simple post processing, such as averaging a series of points in the path through C-space, can yield a more executable path. However, more complex optimization techniques can be computationally expensive and may not always yield shorter path.

On the other hand, modifying search-based motion planning methods so they utilize parallel search schemes that keep track of the C-space which they have visited may be effective for finding shorter, smoother paths than those delivered by our current scheme. Such search schemes limit the amount of redundant work performed because they generate each configuration in C-space at most once. Existing implementations of such methods run into difficulty because they cannot store the C-space they need to solve difficult problems on a single processor. Relatively recent restricted memory schemes such as MA*[4], MREC[17], and PRA*[7], as well as distributed memory schemes such as A* with probabilistic state distribution

[13] appear promising.

4. Conclusion

In summary, we have devised and implemented a parallel robot motion planning algorithm based on quasi best first search with randomized escape from local minima and randomized backtracking on multiple hardware platforms including a 1024 processor nCUBE2, a 544 processor CM-5, and a network of 16 Sun workstations. The method delivers excellent speedup on difficult problem instances, reducing the time required to solve them by more than two orders of magnitude in many cases.

5. Acknowledgements

We would like to sincerely acknowledge Jean Claude Latombe and his group at Stanford University for providing us access to implementations of the Random Path Planner, David Strip and Robert Benner at Sandia National Laboratories for providing us access to the nCUBE2, Bishak Wieckowski for his implementation on a network of workstations, Grama Y. Ananth and George Karypis for their helpful suggestions. We would also like to thank Michael P. Hennessey and Max Donath for assisting us with our 3D implementation.

References

[1] G. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Conf. Proceedings*, pages 483–485, 1967.

[2] S. Arvindam, V. Kumar, V. N. Rao, and V. Singh. Automatic test pattern generation on multiprocessors. *Parallel Computing*, 1991.

[3] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *Int'l Journal of Robotics Research*, 10(6):628–649, 1991.

[4] P. Chakrabarti, S. Ghose, A. Acharya, and S. de Sarkar. Heuristic search in restricted memory. *Artificial Intelligence*, 41:197–221, 1989.

[5] D. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, volume 2, pages 46–51, 1993.

[6] W. Ertel. OR-parallel theorem proving with random competition. In A. Voronokov, editor, *LPAR'92: Logic Programming and Automated Reasoning*, pages 226–237. Springer-Verlag (LNAI 624), 1992.

[7] M. Evett, J. Hendler, A. Mahanti, and D. Nau. PRA^*: A memory limited heuristic search procedure for the connection machine. In *Proc. Third Symposium on the Frontiers of Massively Parallel Computation*, pages 145–149, 1990.

[8] V. Kumar, G. Ananth, and V. Rao. Scalable load balancing techniques for parallel computers. *The Journal of Parallel and Distributed Computing*, (to appear), 1993.

[9] V. Kumar, K. Ramesh, and V. N. Rao. Parallel best-first search of state-space graphs: A summary of results. In *Proc. Nat'l Conf. on Artificial Intelligence*, 1988.

[10] V. Kumar and V. N. Rao. Parallel depth-first search, part II, analysis. *Int'l Journal of Parallel Programming*, 16(6):501–519, 1987.

[11] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishing, Norwell, MA, 1991.

[12] T. Lozano-Pérez. Parallel robot motion planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1000–1007, 1991.

[13] G. Manzini and M. Somalvico. Probabilistic performance analysis of heuristic search using parallel hash tables. Technical report, Scuola Normale Superiore, Pisa, Italy, 1992.

[14] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.

[15] K. S. Natarajan. Expected performance of parallel search. In *Proc. Int'l Conf. on Parallel Processing*, pages III–121–III–125, 1989.

[16] J. Reif. Complexity of the mover's problem and generalizations. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 421–427, 1979.

[17] A. Sen and A. Bagchi. Fast recursive formulations for best-first search that allow controlled use of memory. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, pages 297–302, 1989.

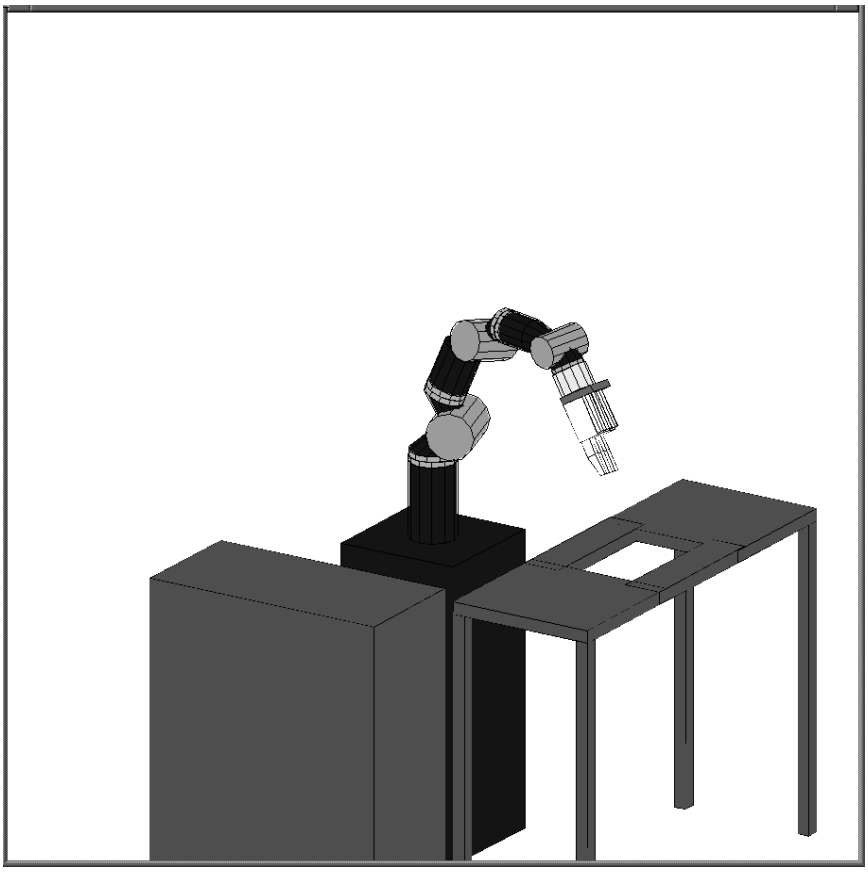

Fig. 6 Accurate scale model of a Seven Degree of Freedom Robotics Research robot operating in a 128 x 128 x 128 cell workspace. Each C-Space axis (joint) has 128 discrete positions.