A hardware architecture for robot path planning.

Pieter P. Jonker , Simon T. Dekker, Ben J.H. Verwer
Faculty of Applied Physics
Delft University of Technology
Lorentzweg 1
2628 CJ Delft, The Netherlands

Abstract

Robot paths can be planned in quantized configuration spaces. Each configuration represents an unique location of the robot. If a matrix of prohibited and permitted transitions is available, paths can be found in the configuration space by heuristic search algorithms. We have studied algorithms and architectures for finding these paths.

We propose a hardware architecture that uses an implementation of the A*-algorithm; a uniform cost search algorithm. Simulations have shown that in a three dimensional quantized configuration space of 32^3 a path can be found in less than 15 ms if 16 processing elements are employed.

Introduction

The problem definition of *robot path planning* is[1]: "Given a moving object, (A), a set of stationary objects, (B_j), at known positions, a legal initial position of A, s, and a legal goal position of A, g, find a sequence of motions that take A from s to g without colliding with any of the B_j." Note that this definition does not include planning of servo actions that minimize the travel time along or close to a predefined path, nor does it include grasp planning or fine motion planning (compliant motions); actions which require 'collisions' in the above mentioned sense.

Collision avoidance is an extension of the path planning problem. An uncertain environment with moving obstacles is assumed.

In this paper a free space method[2] based on quantized configuration spaces[3] is used. A *configuration space* is a space in which each point represents a unique position of a robot in the real world. Each adjacent pair of points in configuration space receives a label, *prohibited* or *permitted*, dependent on whether or not a movement of the robot between those points would result in a collision. However, the transformation of a given robot, (A), and a given set of obstacles, (B_j), into a map of prohibited and permitted transitions is not the subject of this paper. Adjacent points in configuration space are points which coordinates do not differ more than 1 in each dimension. The metric in the configuration space can be chosen to approximate the euclidean distance.[8] Good integer approximations of the euclidean distance can be derived,[12,13]. E.g. for two dimensions, the horizontal and vertical transition costs 5 and the diagonal transition costs 7 lead to a maximum relative error of 4.21% after scaling with 5.71.

Verwer[10] proposed to use a standard heuristic search method in the configuration space: the A*-algorithm.[14,15] We will present here a possible hardware architecture for an implementation of this search algorithm used in quantized configuration spaces, usable for real time robot path finding.

The A*-algorithm

The robot path finding problem naturally falls into the class of graph search problems. The nodes of the graph are points in the N-dimensional configuration space. All nodes are locally connected, except for obstacle points, which are not connected at all (see figure 1).

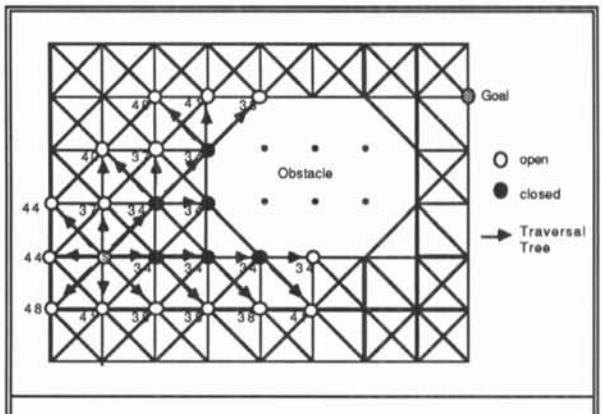

Figure 1. The graph of a 2D path finding process after a few node expansions. The numbers attached to the nodes are evaluation values as computed by an A* type algorithm. The diagonal transition cost is 7, the horizontal and vertical transition costs are 5.

Heuristics information is incorporated by an evaluation function f, which is the sum of two functions g and h. In graph terminology, g(n) represents the costs from the start node s to node n and h(n) represents an estimate of the costs from node n to the goal node g. A* is guaranteed to find the cheapest path if h(n), the heuristic function, is chosen optimistically.

If h(n) is assigned the value zero, the A*-algorithm degenerates into a uniform cost algorithm. In that case waves of equal cost are propagated from the start node in all directions, until the goal node is reached. If h(n) is increased to the maximum still optimistic value d of the costs from node n to the goal node g, the algorithm first expands nodes in the direction of the goal node and later, when obstacles are encountered, nodes in other

directions. We found that the applicability of heuristic information therefore decreased with the complexity of the search space.

The A*-algorithm can be described in 4 steps, see figure 1:

1) Mark the start node s 'open' and calculate an evaluation function f(s).
2) Select the open node n whose f value is smallest. If ties occur, resolve in favour of a goal node.
3) If n is a goal node, mark n 'closed' and terminate the algorithm.
4) Otherwise, mark node n 'closed', calculate f for each successor node of n and mark 'open' each successor node not already marked 'closed'. Remark as 'open' any closed node n_j which is a successor of n for which $f(n_j)$ is smaller now than it was when n_j was marked 'closed'. Goto step 2.

Feasibility of A*- implementations

The calculation most intensive part of the A*-algorithm is the selection of the node with the lowest f value. A sorting algorithm is required here. Bucket sort is the most efficient sorting algorithm, linear in the number of nodes to be sorted.[13,16] It can be used with bounded, integer valued transition costs. The 'open' nodes are stored in buckets. Each buckets has an f value associated with it; nodes with equal f values are stored in the same bucket. The buckets are processed one by one, starting with the bucket with the lowest f value associated with it (in which the start node s is stored).Processing a bucket consists of retrieving each node stored in it and generating successors for each retrieved node. Successors are adjacent nodes which are not yet labelled 'closed'. The f value of the successors is calculated, after which they are stored in the appropriate buckets. A node for which all successors have been generated is labelled 'closed'. Because the maximum difference in f values of generated successors is twice the maximum transition cost, a limited number of buckets suffices.

To obtain a feasible hardware architecture, several implementations of A* have been compared, both theoretically and in simulations. Simulations were performed both on functional level as well as on register transfer level. Four 2-D (size: 256^2) and four 3-D (size: 32^3) test images were used. In the 2-D as well as in the 3-D case the test images consisted of an empty image, an image with a few rectangular constraints, a more complicated constraint image and finally, a maze. The 2-D images are referred to as a,b,c&d; the 3-D images as e,f,g&h. The evaluations and results are reported in [17] and are summarized below:

Both searching from start and goal node simultaneously with using heuristic information decreases the total number of nodes expanded. In the case of using heuristic information (with h(n) the 'as the crow flies' distance to the goal) the savings are highest for simple images. The same applies in the case of employing a bi-directional search. Most important however is that the mechanisms impede each other. Employing a bi-directional search if heuristic information is used degrades the performance, relative to the case were only heuristic information is used.

When using directional information, a priori pruning of the search graph is possible. The savings consist of a reduction in the mean number of successors. Normally, nodes are opened many times before they receive the label 'closed'. If directional information is used this inefficiency diminishes. All nodes adjacent to a generating node that can be reached via a cheaper path from the node which generated the generating node (see figure 2) are not considered as successors. Of course one has to check if the cheaper path is not prohibited.

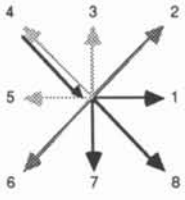

Figure 2. Directional information can eliminate successors. Suppose the central node was generated from neighbour 4. Neighbour 4 does not need to be considered as successor, neighbours 2,3,5 and 6 only if the transitions from neighbour 4 to neighbours 3 and 5 or to neighbours 2 and 6 via neighbour 3 and 5 are prohibited.

Simulations showed that this method (DIR) yields a 50% reduction in the 2D and 3D test images, but savings diminish in higher dimensions. An adjusted method (SDIR) which allows only smooth direction changes (1,7 & 8 in figure 2) exhibit increasing savings in higher dimensions but might miss the solution path if the constraint image contains 1 pixel thick passages.

A postiori pruning of the search graph is also possible if only the successor that is 'better' (in terms of 'best evaluation value so far') than the previously generated successor to the same node will be inserted in 'open'.

Hardware

Due to the evaluation of the functional simulations it was decided to focus on an architecture for the A* algorithm using bi-directional search, a priori pruning methods DIR or SDIR, a postiori successor pruning and (optional) heuristics.

An obvious source of parallelism in the A* algorithm is the wave front of equal cost that has to be expanded. It was decided that the path finding machine (PFM) should consist of identical processing elements (PEs), in which each PE handles a number of 'open' nodes. Moreover, the PEs should only have access to a mutually disjoint subset of the large data structures. We will elaborate on this later. Each PE can access the data on only a subset of the nodes in the search graph. Communication between the PEs remains necessary to exchange successor nodes. Figure 3 shows the architecture of the PFM consisting of a

number of PEs, a data switch network and a single global controller .

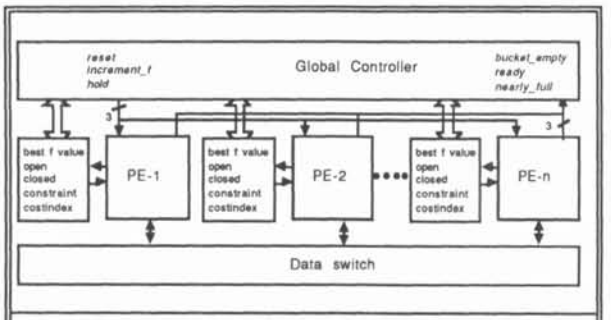

Figure 3. The architecture of the Path Finding Machine.

The **global controller** synchronizes the PEs and might be implemented on a conventional sequential processor. The input and outputs of a PE are *reset, increment_f, bucket_empty* , *ready* and *nearly_full*. During the actual search process, the global controller has only a passive job, which can even be implemented with some logic. If *reset* is asserted all PEs clear their current evaluation value f. After reset, each PE starts to process the nodes in the bucket indicated by their local f value. When a PE is ready with this bucket it asserts the *bucket_empty* signal. The PEs may however proceed with the next buckets, (f+1), (f+2), ... until the bucket number exceeds f+d; d being the minimal transition cost (e.g. 5). Note that proceeding is allowed since buckets with index f generate successors with an evaluation value of at least f+d. The global controller waits until all PEs assert the *bucket_empty* signal. At that moment, *increment_f* is asserted and all PEs are enabled to proceed. If some PE detects the end condition, it sends the *ready* signal to the global controller, which then terminates the search. The global controller 's active tasks are initialization and finding the final solution path.

The **data switch network** transfers successor node data between PEs and can be implemented for a modest number of PEs with a synchronous data bus.

If the PFM consists of more than a modest number of PEs the switch is more difficult to implement and will become the engines bottle-neck. A variant of the token ring approach as used in the Manchester data flow system[18] and the IMage Pipeline Processor from NEC [19] can be used. Since the successor nodes (the tokens) have a fixed structure, and a delay between sending a node and receiving this node does not matter, this approach is applicable. PEs are connected in a local ring, see figure 4. The elements labeled 'S' are token switches. If a token on the global ring enters such a token switch, it is checked whether an address field in the token matches the address of the current PE. If so, the token enters the PE; if not the token is sent to the next 'S' element. Tokens in the global ring have a higher priority than tokens produced by the PEs.

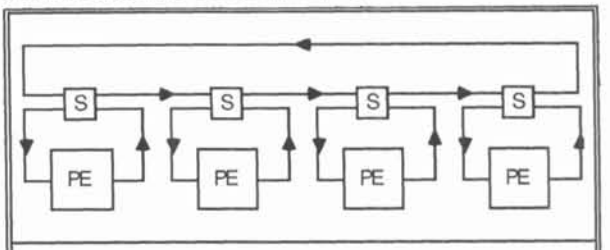

Figure 4. A token ring as interconnecting network for a modest number of PEs.

To improve the performance for systems with a large number of PEs, the 'S' element can be used to connect different rings together. A candidate interconnecting structure is a hypercube, see figure 5. An N-dimensional cube can be constructed by inserting N 'S' elements in the ring. E.g. a 256-PE path finding engine could consist of 64 rings of 4 PE's, where each 4-PE ring has 6 'S' elements to connect it with other rings.

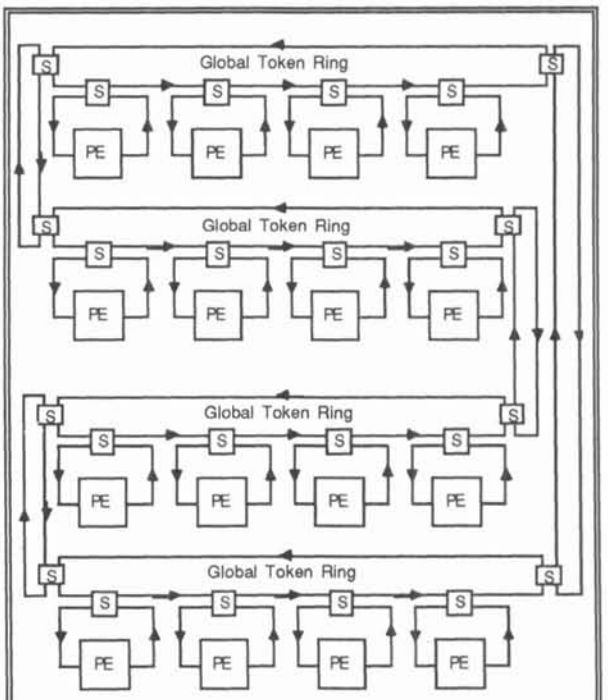

Figure 5. Connection of 4 PE rings in a 2D hypercube.

A draw back of the distributed computing architecture is that parallelism cannot be forced; we must hope that the nodes in the buckets are distributed uniformly over the PEs. The distribution of the nodes of the search graph (the wave front) over the PE's is a fixed, fine grained, spatial distribution, and is implicitly executed by the coordinate mapping function. This function assigns a PE number to a coordinate. It was decided to use a simple mapping function based on the binary representation of the coordinates: The least significant bits of the coordinates are used as PE address. Simulations showed that the total number

of input nodes in 'open' was divided almost uniformly over the PEs, with as notable exception one of the 8 test images; a generated 3D maze, which was too regular to achieve an even distribution.

Figure 6 shows the architecture of a **processing element**. Each PE is a synchronous pipeline that handles, enabled by the global controller, a number of open nodes. Each clock cycle every PE generates a successor node, which is either stored in its own input FIFO or over the network in the FIFO of another PE.

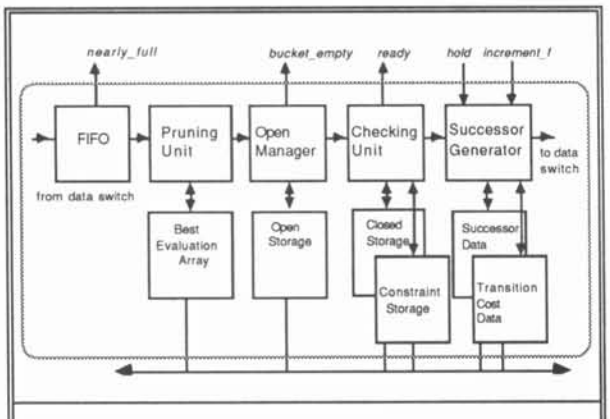

Figure 6. The architecture of a Processing Element.

The PE consist of a number of pipelined functional units, each with memory for data and parameters. The individual tasks of those units are listed below:

INPUT_FIFO
On the average, the OPEN manager processes input nodes at the same speed as the successor generator creates them. However, the momentary node rate can be higher, since at times successors from different PEs can all be sent to the same PE. The FIFO generates the *nearly _full* signal, which can be used to temporarily hold the successor generators of other PEs.

PRUNING_UNIT
Uses a 'best evaluation array' or 'distance image', which keeps for every node coordinate the best evaluation value so far. Only nodes that are 'better' than the best evaluation value so far may pass. The array can also be used as traversel tree by the global controller to find the final solution path.

OPEN manager
The administrator of the local 'open' data structure, organized in buckets. Incoming nodes are stored in the bucket indicated by its f value. Out comming nodes are retrieved from bucket $\{f_X|(f \leq f_X < f+d)\}$. The *bucket_empty* signal is generated by this block.

CHECKING unit
The unit interacts firstly with the CLOSED array. It checks whether the input node is an element of 'closed'. If the node was an element of 'closed', it is discarded. If not, it checks whether the end condition is met. Then the *ready* signal is asserted. If not, it is checked whether the node is an element of the CONSTRAINT image; then it is discarded. Optionally it has an interface with some external constraint checking hardware (e.g. for moving objects).

SUCCESSOR generator
This unit actually expands a node. The full coordinate must be restored in this unit from the local coordinate and the PE number. Depending on a direction field of the node, successors can be generated from a table, which can be filled with DIR- or SDIR types of successors. The new evaluation value is generated for each successor. Optionally a cost index storage can be connected containing a cost index for each local node coordinate. (To allow space dependency of costs; e.g. transitions near objects can be given higher costs.) The successor generator has a heuristic unit to calculate the heuristic function h.

Simulations.

For the register transfer simulation of the PFM, each of the units was further refined and the following assumptions were made:
All units of the PE are separated with pipeline registers and a two phase clock is used: (Ø1;compute and Ø2;transfer). All units can be constructed in such a way that the processing time needed for a for a compute step is twice the cycle time of the DRAMS used for the tables. Hence a realistic cycle time is 400ns. All units take one elementary cycle, except for the successor generator which takes one cycle per generated successor. If a heuristic function is used, two cycles per successor are necessary.
The simulations have been run for 8 test images {a...h} using all combinations of unidirectional / bi-directional search, pruning / no pruning and heuristics / no heuristics . A priori pruning was not applied, since the benefits of this were discovered later on. The used distance coefficients or transition costs were 5,7 for 2D images and 7,10,12 for 3D images. Table 1 gives the data of a typical simulation.Table 2 gives the processing times of the path finding machine, using no heuristics, no a postiori pruning and bi-directional search.

The software version (in C) of the algorithm using image e, ran in 23s on a 68000,12Mhz machine. The Path Finding Machine takes 0.01 sec for this image. A speed up of 2300.

Conclusions

Robot path planning in a quantized configuration space offers prospects for real-time applications. It has been shown that a path in a 32^3 configuration space can be found in less than 15 ms with 16 PEs, if the matrix of prohibited and permitted transitions is known. Future research will focus on algorithms and architectures for establishing the transition possibilities. Possible other application areas like VLSI routing and AI techniques will be investigated to generalize the approach.

The hardware will be implemented in CMOS VLSI technique and be applied in the Delft Intelligent Assembly Cell.

Table 1. Typical simulation output for the PFM: For a 1,2 and 8 PE system, the number of cycles, the FIFO depth, the number of nodes in 'open', the number of nodes after checking, the idle- and blocked cycles of the checking unit and the idle cycles of the successor generator are given.

PEs	CYCLES	PE #	FIFO depth	open nodes	checking unit idle	checking unit blocked	checking unit nodes	succ. Gen. idle cycles
1	275129	0	1	200644	4	77363	41030	72213
2	126027	0	13	99865	4	27148	20507	40404
		1	4	84608	15	43035	20396	25081
8	30146	0	11	22376	312	7714	5124	5655
		1	16	21480	313	8848	5068	5336
		2	10	21943	208	8271	5143	5504
		3	13	21748	139	8722	5089	5237
		4	8	23889	2313	4130	5117	10189
		5	12	23485	964	5977	5109	7692
		6	9	24443	1824	4088	5130	8954
		7	12	23851	549	6032	5129	7334

Table 2. Processing times of a simulated PFM. The processing times are based on clock cycles of 400 ns.

image	1 PE	16 PE's	speed up
a	131.9 ms	9.5 ms	13.8
b	110.1 ms	6.7 ms	16.4
c	158.4 ms	9.1 ms	17.4
d	103.5 ms	7.8 ms	13.2
e	192.2 ms	10.1 ms	19.0
f	246.6 ms	13.9 ms	17.7
g	17.5 ms	1.2 ms	14.1
h	9.7 ms	1.5 ms	6.5

Acknowledgement

This research was supported by the Dutch Government as a part of the SPIN-FLAIR-DIAC project.

References

(1) T. Lozano-Pérez, "Robot programming", in AI in the 1980's and beyond, W.E.L. Grimson and R.S. Patil, ed., MIT Press, 1987.

(2) K.S. Fu, R.C. Gonzalez and C.S.G. Lee, Robotics, Control, Sensing, Vision and Intelligence, MacGraw-Hill, 1987.

(3) T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach", IEEE Trans. Comput., vol. C-32, No. 2, Feb. 1983.

(4) W. Gilles, A universal computerprogram for the determination of a collision free path for an industrial robot, graduation thesis, Delft University of Technology, Faculty of Mechanical Engineering, The Netherlands, 1984.

(5) C.J. Lee, An algorithm for path connections and its applications", IEEE Trans. on Elec. Comput., pp 346-365, Sep. 1961.

(6) W.E. Howden, "The Sofa Problem", Comput. J., vol. 11, pp. 299-301, Nov. 1968.

(7) P.W. Verbeek, L. Dorst, B.J.H. Verwer and F.C.A. Groen, "Collision avoidance and path finding through constrained distance transformation in robot state space", in Proc. of Intelligent Autonomous Systems, Amsterdam, Dec. 8-11,1986.

(8) G. Borgefors, "Distance transformations in digital images", Computer Vision, Graphics and Image Processing, vol. 34, pp. 344-371, 1986.

(9) K.G. Shin and N.D. McKay, "Selection of near-minimum time geometric paths for robotic manipulators", IEEE Trans. on Automatic Control, vol. AC-31, no. 6, pp. 501-511, June 1986.

(10) B.J.H. Verwer, "Heuristic search in robot configuration space using variable metric", NASA Conference Publication 2492, presented at the 3rd Conf. on Artificial Intelligence for space applications, Nov. 2-3, Huntsville, Alabama, 1987.

(11) M. Yamashita and T. Ibaraki, "Distances defined by neighborhood sequences", Pattern Recognition, vol. 19, no. 3, pp 237-246, 1986.

(12) S.T. Dekker, "Toward a hardware architecture for robot path finding", graduation thesis, Delft Univ. of Technology, 1987.

(13) B.J.H. Verwer, P.W. Verbeek and S.T. Dekker, "An efficient uniform cost algorithm applied to distance transforms", to appear in IEEE Trans. on Pattern Analysis and Machine Intelligence.

(14) P.E. Hart, N.J. Nilson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", IEEE Trans. on Systems Science and Cybernetics, vol. SSC-4, no.2, pp. 100-107, 1968.

(15) J.Pearl, Heuristics, Addison Wesley, 1985., ISBN 0-201-05594-5.

(16) A. Aho, J. Hopcroft and J. Ullman, The design and analysis of computer algorithms, Addison Wesley, 1974.

(17) Robot Path Planning by Heuristic Search, B.J.H.Verwer, S.T.Dekker, P.P.Jonker, F.C.A.Groen, accepted for the 12th EMACS World Congress on Scientific Computation, PARIS, july 18-22, 1988.

(18) J.R.Gurd, C.C.Kirkham, I.Watson, The manchester prototype dataflow computer, Communications of the ACM, vol 28-1, Januari 1985, pp34-52.

(19) M. Iwashita et al., A data driven VLSI image processor (IMPP). In: I. Uhr, K. Preston, S. Levialdi, M.B.J. Duff Evaluation of Multi computers for Image Processing, Academic Press Inc., Orlando Florida USA, 1986.