COMPUTATIONAL COMPLEXITY OF PATH PLANNING ALGORITHMS BASED ON SAFE TRIANGLES AND QUADTREE AS WORK SPACE REPRESENTATIONS: A COMPARISON†

Xiaozhao Mao

Institute of Control Engineering, Section Control Systems Theory and Robotics, Technical University Darmstadt, Schlossgraben 1, D-6100 Darmstadt, Germany

Abstract. Computational complexity of two path planning approaches for a two-dimensional (2D) work space is considered: The approach using safe triangles and the approach based on quadtree. The average cost of establishing work space representation, i.e. of building the safe triangles and the quadtree respectively, are estimated. Both approaches are simulated to plan paths for numerous work spaces with obstacles which are generated at random. The results of the simulation are summarized to show the expected behavior of each approach. Comparison based on analyses and simulation is presented.

Keywords. Computational complexity; path planning; robotics; space representation.

INTRODUCTION

Automatic planning of collision free paths among obstacles is essential for autonomous mobile robots. The basis of path planning is a work space representation describing the free and/or the forbidden volume in the work space so that possible paths are summed up into a graph. The finding of an appropriate path is performed by searching the graph under minimizing a given cost criterion. The efficiency of a path planning algorithm depends tightly upon the work space representation it uses. It is of both theoretical and practical interest to compare the efficiency of different algorithms in order to disclose the advantages of one algorithm versus the other.

In this paper the computational complexity of two path planning approaches for 2D work space are compared: one using safe triangles [Mao, 1990] and one based on quadtree [Kambhampati and Davis, 1986; Soetadji, 1986]. Throughout the paper the following abbreviations are used:

QT:	quadtree;
QT-R:	QT-representation;
Q:	square in QT-R;
FQ:	free square in QT-R;
QTG	search graph for QT-R;
DS:	distance segment;
PDS:	passable DS;
ST:	safe triangle;
ST-R:	ST-representation;
STG:	search graph for ST-R;
C[x(·)]:	complexity of operator x(·);
C_{qt}	expected complexity of constructing QT-R;
C_{st}	expected complexity of constructing ST-R;
N:	number of obstacles in work space;
n_i:	number of edges of the i-th obstacle;
n:	number of all obstacle edges.

Additional notations will be introduced in context.

In ST-R, the obstacles and the robot are described by convex polygons. The shortest line segment connecting two obstacles is called distance segment DS between the obstacles. We consider only the DSs which are longer than the minimum cross section of the robot and do not intersect any obstacles. Moreover, if two DSs intersect each other, the longer one is taken away from consideration. The remaining DSs, called passable distance segments PDSs, allow the robot to pass through safely between the corresponding obstacles. For each PDS two STs, one on each side of the PDS, are constructed, as illustrated in Fig. 1. Additionally, a set of allowed orientations for the robot, determined by the length of the PDS, is related to the STs. These STs enclose the safe positions for the robot before and after the passage, as long as its orientation stays in the allowed orientation set. The search graph STG consists of all STs as nodes. If two STs overlaps each other as well as their orientation sets, the corresponding nodes in STG are connected by an arc.

In QT-R, the robot is considered to be a point. To describe QT-R the layer concept suggested by Adolphs (1988) is used. The work space is normalized to a 1x1 square which is the layer 0 of QT. It is divided into 4 equal subsquares forming layer 1 of QT. Each square Q_j on layer j is checked to indicate whether it is free or occupied by obstacles. If Q_j is partially occupied, it is divided into 4 equal subsquares again which belong to layer j+1, as illustrated in Fig. 2. This procedure continues until layer s, the minimum quantization layer, is arrived at. The free squares FQs of all layers build the nodes of the search graph QTG. The arcs connecting the nodes indicate the neighborhood of corresponding FQs in the work space.

Normally, the complexity of an algorithm is analyzed for the worst case [Lee and Preparata, 1984]. In our case, however, the worst case analysis seems to be unsuitable. In fact, the worst case would mean for QT-R that all squares on all layers are partially occupied. This would mean an unpassable environment. Therefore the comparison is made for expected behavior.

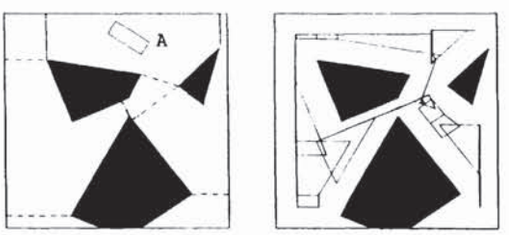

Fig. 1 ST-R: an example

†This work was supported by DFG (Deutsche Forschungsgemeinschaft)

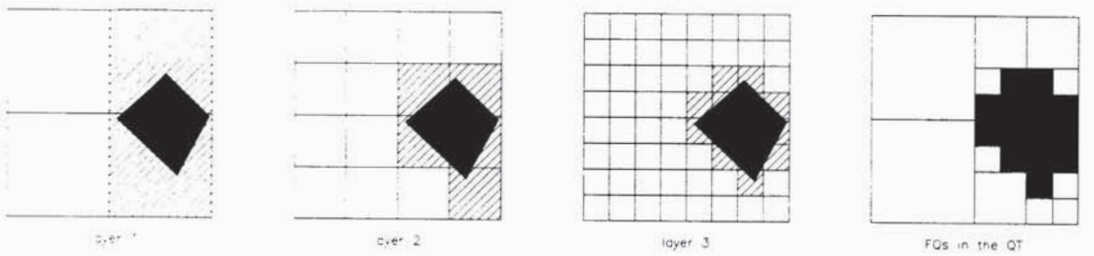

Fig. 2 Quadtree – layer concept

In addition to analytical discussions results of computer simulations are presented to illustrate the efficiency of each algorithm. The work spaces used by simulations were normalized to 1x1 squares. The obstacles in a work space are generated as follows:

1) generate $n_{max} \geq n$ points in the work space randomly;
2) divide the work space with randomly generated straight lines into N subregions so that each subregion contains at least 3 points.
3) for each subregion an obstacle is constructed by building the convex hull of the points in the subregion [Preparata and Hong, (1977)].

A rectangle of 0.08x0.05 was taken as the robot for ST-R. The illustrations represent the average of simulation results for numerous work spaces with same parameters labeled in the figures. The simulations were run on a PC-AT/286. To give a relative scale the CPU time t used by an operation is converted to equivalent flops EF by the formula

$$EF = t/(t_{\text{vector addition}} + t_{\text{dot product}})$$

COMPLEXITY OF ST-R

A ST-R is constructed in two steps: determining the PDSs and building STs. The DS between two convex polygons with n_{i1} and n_{i2} edges respectively can be determined in $O(n_{i1}+n_{i2})$ time [Gilbert and co-workers, 1988]. The total number of edges considered while determining all N(N-1)/2 DSs can be given by

$$\sum_{j=2}^{N}[(j-1)n_j + \sum_{i=1}^{j-1} n_i] = \sum_{j=2}^{N}(j-1)n_j + \sum_{j=2}^{N}\sum_{i=1}^{j-1} n_i$$

$$= \sum_{j=2}^{N}(j-1)n_j + \sum_{j=1}^{N-1}(N-j)n_j = (N-1)n.$$

Hence it is a O[(N-1)n] operator to determine all DSs. Checking for intersections of DSs can be done in $O(L \cdot \log_2 L)$ time [Shamos, 1976] with

$$L = N(N-1)/2,$$

$$L \cdot \log_2 L = \frac{N(N-1)}{2} \cdot \log_2 \frac{N(N-1)}{2}$$

$$\approx N(N-1)(\log_2 N - 1/2).$$

Whether a DS crosses at least one obstacle can be determined in O(N) time, i.e. it is a $O[N^2(N-1)/2]$ operator to check all DSs. Summarizing, it turns out that PDSs can be determined in

$$O[(N-1)n] + O[N(N-1)(\log_2 N - 1/2)] + O[N^2(N-1)/2]$$

time. The last two terms above correspond to the worst case where all DSs are passable. Since this is unlikely in the usual case N>3, the actual complexity will not be dominated by them. Indeed, the larger N is, the more possible it is that a DS intersects either another DS or an obstacle and hence is taken away. Figure 3 illustrates the average time t_dist used in simulation for determining PDSs. It is linear to n/N for constant N and quadratic to N for constant n/N, what suggests the expected complexity of determining PDSs to be

$$O(N^2 \cdot n/N) = O(N \cdot n). \quad (1)$$

The operator of building STs for a PDS is linear to N [Mao, 1990]. Since there can be at most N(N-1)/2 PDSs, it will take at most $O[N^2(N-1)]$ time to building all STs. With the same argument as above it is reasonable to assume that the average number of STs is much less then N(N-1) and hence the expected complexity of building STs is lower then a cubic function of N. The simulation confirms this assumption: The average time t_tr taken by building STs is nearly linear to N (Fig.4, Fig. 5). The average time t_st used by constructing ST-R in simulation is presented in Fig. 5. It shows that the expected complexity C_{st} of constructing ST-R can be estimated by

$$C_{st} = O(N \cdot n) + O(N) \quad (2)$$

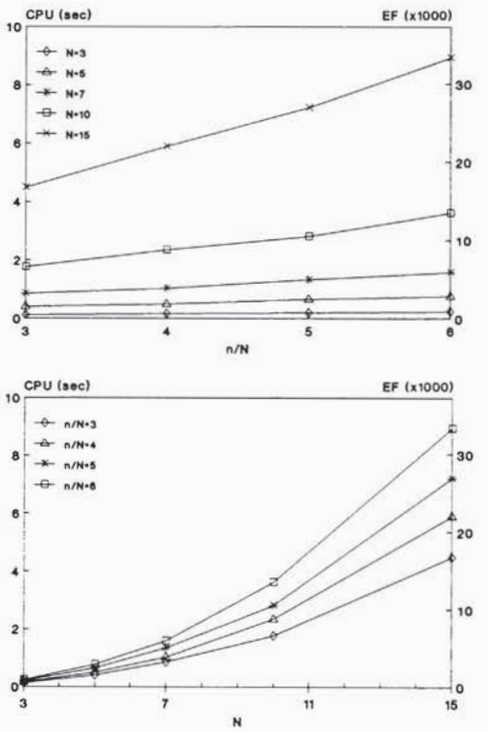

Fig. 3 Time for determ. PDS (t_dist)

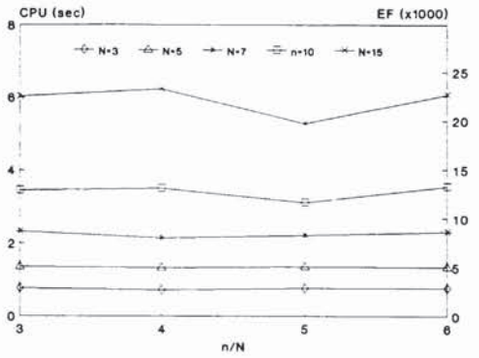

Fig. 4 Time for building STs (t_tr)

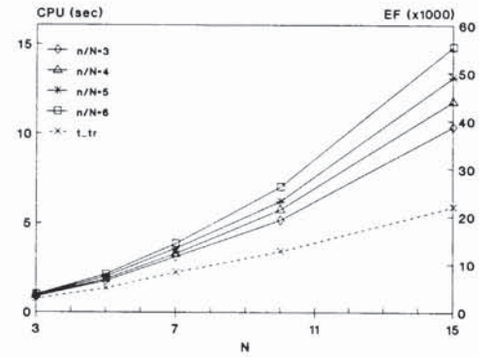

Fig. 5 t_st (=t_dist+t_tr)

COMPLEXITY OF QT-R

For building a QT-R it is necessary to check whether a square Q_j on layer j is occupied by an obstacle. The efficiency of building QT-R depends directly on the check algorithm. In this paper the following check algorithm is used:

if	one obstacle edge intersects Q_j
than	Q_j is partially occupied;
else if	the center point of Q_j is inside an obstacle
than	Q_j is totally occupied;
else	Q_j is free of obstacles.

The algorithm consists of two checks: **intersect**(e,Q) checking whether an obstacle edge e intersects a square Q, i.e. whether e has at least one interior point of Q, and **inside**(P,e) checking whether a point P is on the "inner" side of e. The first check is carried out for obstacle edges one by one until either an edge intersecting Q is encountered or all edges are checked. The second check is activated only if **intersect**(e,Q) is negative for all edges. In this case, if Q is free, **inside**(P,e) will be performed for all obstacles, otherwise the number of obstacles checked by **inside**(P,e) depends on the order of the obstacles at checking. Suppose that in average N/2 obstacles are checked before the obstacle containing Q is encountered and for each obstacle that does not contain Q the half of its edges are checked. The average number of carrying out **inside**(P,e) equals $p_{in}\cdot n/4+(1-p_{in})\cdot n/2$, p_{in} being the probability that Q is inside an obstacle.

Let m_j be the expected number of squares on layer j which are partially occupied, k_j and K_j the expected number of carrying out **intersect**(e,Q_j) and **inside**(P,e) respectively for checking Q_j, and p_j the probability that e intersects Q_j. The expected complexity C_{qt} of constructing QT-R can be given by

$$C_{qt}=\sum_{j=1}^{S} m_{j-1}\cdot\{C[\mathbf{divide}(Q)]+4k_jC[\mathbf{intersect}(e,Q_j)] +4c_jK_jC[\mathbf{inside}(P,e)]\} \quad (3)$$

with

divide(Q): divide Q into 4 equal subsquares;

$$m_0=1; \qquad m_j=4p(j)m_{j-1}, \qquad j\geq 1; \quad (4)$$

$$k_j=\sum_{k=1}^{n-1}[(1-p_j)^{k-1}p_jk]+(1-p_j)^{n-1}n=\frac{p(j)}{p_j}; \quad (5)$$

$$p(j)=1-(1-p_j)^n;$$

$$c_j=(1-p_j)^n;$$

$$n/4\leq K_j\leq n/2.$$

Apparently the factor in the braces of Eq. (3) is linear to n, indicating C_{qt} is of O(n) for constant s. Simulation for s=7 shows that t_qt, the average time needed for constructing QT-R, is nearly linear to n/N for constant N and nearly linear to N for constant n/N (Fig. 6). Hence,

$$C_{qt}=O(N\cdot n/N)=O(n) \quad (6)$$

is a reasonable estimation of C_{qt} for constant s. To reveal the influence of s on C_{qt} the number m_j has to be enumerated which depends on the probability p_j. Let q_j be the expected number of squares on layer j which are intersected by e, p_j can be calculated by

$$p_0=1;$$

$$p_j=q_j/4m_{j-1}=\frac{1}{4}\cdot\frac{q_j}{q_{j-1}}\cdot\frac{p_{j-1}}{p(j-1)} \qquad j\geq 1. \quad (7)$$

To estimate q_j let l be the average length of e, and

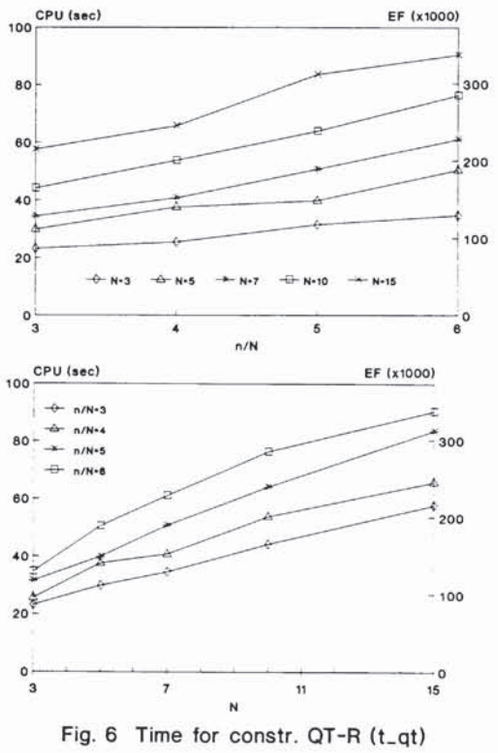

Fig. 6 Time for constr. QT-R (t_qt)

$$l_j = l/2^{-j} = 2^j \cdot l \qquad (8)$$

be the relative length of e on layer j. If we ignore the zero probability event that e just overlaps a dividing line, the number of squares intersected by e is distributed in the interval I_j

$$I_j = [\text{int}(l_j\sqrt{2}/2),\ \min(\text{int}(l_j\sqrt{2})+2,\ 2^{j+1}-1)]$$

with int(x) as the smallest integer not less than x

$$\text{int}(x)-1 < x \le \text{int}(x).$$

In the following considerations we assume $\text{int}(l_j\sqrt{2}) \le 2^{j+1}-3$. For simplicity the distribution is assumed to be uniform, i.e. the probability p(u,j) that e just intersects u squares on layer j is given by

$$p(u,j) = \begin{cases} \dfrac{1}{\text{int}(l_j\sqrt{2})+3-\text{int}(l_j\sqrt{2}/2)} & u \in I_j. \\ 0 & u \notin I_j. \end{cases} \qquad (9)$$

It follows

$$q_0 = 1;$$

$$q_j = \sum_u [p(u,j)\cdot u] = \frac{\text{int}(2^j l\sqrt{2})+\text{int}(2^{j-1} l\sqrt{2})+2}{2}$$

$$\approx \frac{2^j l\sqrt{2}+2^{j-1} l\sqrt{2}+3}{2} \qquad j \ge 1. \qquad (10)$$

It can be verified by using Eq. (7) and Eq. (10) that $p_j \ge 1-(3/4)^{1/n}$. Using this relation in Eq. (4) yields:

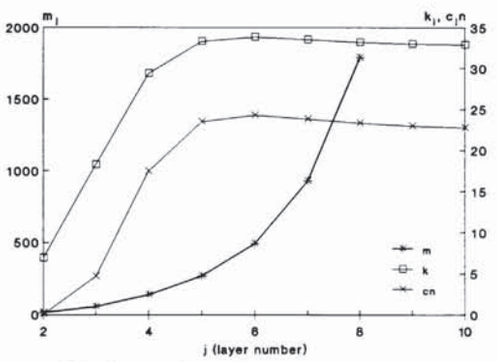

Fig. 7 m_j, k_j and $c_j n$ on each layer

$$m_j/m_{j-1} = 4p(j) = 4\cdot[1-(1-p_j)^n] \ge 1. \qquad (11)$$

Calculations with l and n as parameters point out that m_j/m_{j-1} always approaches 2 as j increases. Figure 7 shows m_j in comparison with k_j and $c_j n$ for a work space with 15 triangular obstacles and l=0.14: while k_j and $c_j n$ tend to become constant, m_j increases exponentially. Accordingly, the time needed for constructing QT-R depends exponentially on s (Fig. 8). In the simulations s was set to be 7 if not said otherwise.

COMPARISON

Work space representation

The complexity of constructing ST-R is $O(N\cdot n)+O(N)$ in average, while that for QT-R is linear to n. This, however, does not mean that constructing ST-R must take more time than constructing QT-R. To the contrary, simulations demonstrate that ST-R needs essentially less time than QT-R does (Fig. 5, Fig. 6). This is because huge number of squares have to be treated during constructing QT-R and therefore the coefficients of Eq. (3) dominate C_{qt}. Additionally, due to its exponential dependence on s C_{qt} is influenced sensitively by the number of layers in a QT.

Findpath

The findpath problem is solved by searching the graph with a search algorithm. A comparison of searching time taken by a search algorithm for different work space representations, however, does not always say much about efficiency of each work space representation, because search algorithm behavior depends strongly on the structure of the search graph, the start and goal positions, and the heuristic used at searching which changes usually with the designer. Therefore, the comparison is concentrated on the size of the search graph, i.e. on the number of nodes in the graph as it describes implicitly the potential complexity of searching.

Though both the average number of STs and that of FQs increase somehow linearly to N (Fig. 9), the number of FQs is much greater than that of STs. This indicates that QTG is in general much greater than STG and hence much more difficult to search. Indeed, while the well known A*-algorithm searches STG efficiently, it does not work well for QTG because the OPEN-list grows quickly so that ordering of it becomes tedious. To avoid this difficulty Adolphs (1988) suggested to use a controlled depth-first algorithm for QTG. In this algorithm only the direct descendants of the current node are considered for determining the next node to be expanded at each search step, instead of considering the whole OPEN-list as the A*-algorithm does [Nilsson,1982]. In Fig. 10 the average time t_srch used by the A*-algorithm for STG and by the controlled depth-first algorithm for QTG, respectively, are illustrated. In contrast with the nearly linear dependence of t_srch on N for STG, t_srch for QTG scatters irregularly as a local search algorithm usually behaves. It is shown that searching in STG seems to take more time than searching in QTG as N in-

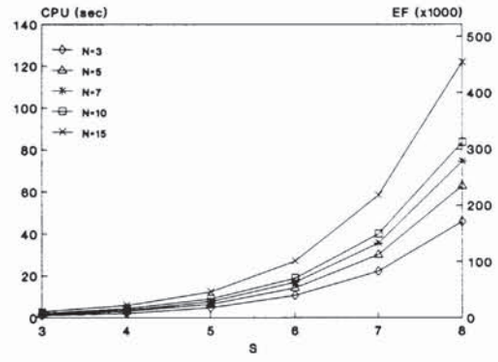

Fig. 8 t_qt depending on s (n/N=3)

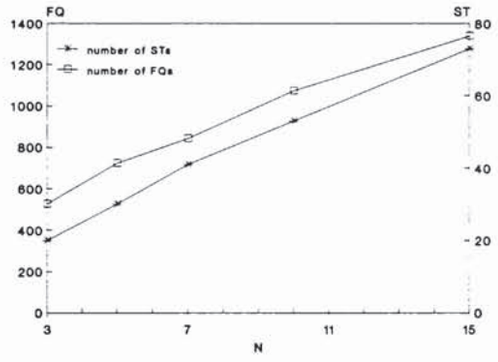

Fig. 9 number of safe regions

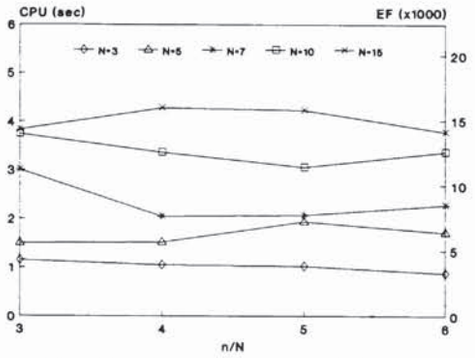

Fig. 10a Search time t_srch in STG

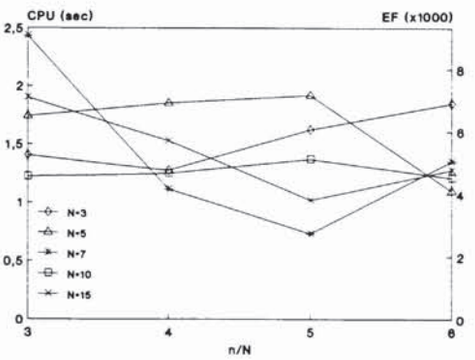

Fig. 10b Search time t_srch in QTG

creases. This is because that, on the one hand, the robot was considered in QT-R as a point and hence the search problem was simplified by ignoring the effect of the robot extent. On the other hand, while the neighbors of a square in QT can be determined trivially in the layer concept [Adolphs, 1988], the connecting arcs in STG are more complicated to determine since it has to be checked whether two STs as well as their orientation sets overlap each other. Therefore, to expand a node in QTG is easier than that in STG.

Remarks

Since the findpath based on QT-R can only be carried out for point formed robot, it is necessary to grow the obstacles so that the robot can be treated as a point, before QT-R is constructed. This can be done by transform the work space into the C-space [Lozano-Perez, 1981, 1983]. However, since in general a convex polygon of n_i edges becomes one of n_i+n_0 edges in C-space for a robot with n_0 edges, the number of total obstacle edges is increased to $n+Nn_0$ after transformation. This will raise C_{qt} as discussed before. In addition, if the orientation change of the robot is taken into account by using θ-slices [Lozano-Perez, 1983], the complexity of both constructing QT-R and searching QTG will increase drastically because one QT for each θ-slice has to be constructed and the QTG has to include FQs in all QTs.

CONCLUSION

Constructing ST-R and QT-R are both operations depending not only on the numbers N and n, but also on form and location of each obstacle. Hence their computational complexity is difficult to analyze. In this paper the expected complexity C_{st} and C_{qt} were considered. It was disclosed that C_{st} is linear to $N \cdot n$ and that C_{qt} is linear to n and exponential to s. A formula for calculating C_{qt} was established.

Quantitative comparison based on simulations showed that in average ST-R might be more efficient than QT-R for path planning, especially when the orientation of the robot has to be considered. However, since ST-R is a rather conservative work space model, it does not work well in clustered work spaces where few STs overlap each other, while QT-R can always work by increasing the number of layers.

REFERENCES

Adolphs, P. (1988). Umweltmodell und Bahn-plannung von Robotersystemen im Raum. Internal Report, Inst. of Control Eng., Techn. Uni. Darmstadt.

Gilbert, E.G., Johnson, D.W., and Keerthi, S.S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE J. of Robotics and Automation, vol. 4, no.2, pp. 193-203.

Kambhampati,S., and L.S.Davis. (1986). Multiresolution path planning for mobile robots. IEEE J. of Robotics and Automation, vol. RA-2, no.3, pp. 135-145.

Lee, D.T., and Preparata, F.P. (1984). Computational geometry-A survey. IEEE Trans. on Computers, vol. C-33, no. 12, pp. 1072-1101.

Lozano-Perez,T. (1981). Automatic planning of manipulator transfer movement. IEEE Trans. on Systems, Man, and Cybernetics, vol. SMC-11, no.10, pp. 681-698

Lozano-Perez,T. (1983). Spatial planning: A configuration pace approach. IEEE Trans. on Computers, vol. C-32, 108-120.

Mao, X. (1990). Concept of path planning with safe triangles as safe space representation. Preprints of IMACS-IFAC Int. Symp. MIM-S² 90, Brussels, issue no. II.C.5.

Nilsson,N.J. (1982). Search strategies for AI production systems. In Principles of Artificial Intelligence, Chapter 2. Springer Verlag. 53pp.

Preparata, F.P., and Hong, S.J. (1977). Convex hulls of finite sets of pointers in two and three dimensions. Commun. ACM, vol. 20, no. 2, pp. 87-93.

Shamos, M.I. (1976). Geometric intersection problems. Proc. 17th IEEE Annu. Symp. on Found. Comput. Sci., pp. 208-215.

Soetadji,T. (1986). Cube-based representa-tion of free space for the navigation of an autonomous mobile robot. Preprints of An Int. Conf. on Intelligent Autonomous Systems, Amsterdam, pp. 546-561.