RISC-Linz
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria, Europe

Preparations With Workspace Information For Meaningful Robot Motion Planning in Configuration Space.

Carl Van GEEM

(March 16, 1995)

RISC-Linz Report Series No. 95-14

Editors: RISC-Linz Faculty
E.S. Blurock, B. Buchberger, C. Carlson, G. Collins, H. Hong, F. Lichtenberger, H. Mayr, P. Paule, J. Pfalzgraf, H. Rolletschek, S. Stifter, F. Winkler.

Supported by: Austrian Ministery for Science and Research, project PARAGRAPH.

Copyright notice: Permission to copy is granted provided the title page is also copied.

Preparations With Workspace Information For Meaningful Robot Motion Planning in Configuration Space.

Carl Van Geem
RISC-Linz
Research Institute for Symbolic Computation
Joh. Kepler University
A-4040 Linz, Austria
Internet: Carl.Van.Geem@risc.uni-linz.ac.at

March 1995

Abstract

Robot Motion Planning using the Configuration Space (Cspace) consists of two subproblems: the construction of Cspace and planning a path for a point in Cspace. In this paper we firstly tackle the construction of Cspace. In particular we want to have an algorithm that computes a non-uniform grid on a 6 dimensional Cspace. We describe an algorithm for computing a non-uniform grid for a robot arm with 6 rotational joints, with links modelled by cylinders. Secondly we discuss several strategies for the construction of a potential field on the non-uniform grid. Finally we describe operations on the grid and on the potential field necessary when the position of an obstacle changed in Workspace.

1 Specification of the Problem.

Robot Motion Planning using the Configuration Space (Cspace) consists of two subproblems: the construction of Cspace and planning a path for a point in Cspace. In this paper we consider the problem of the construction of a non-uniform grid on Cspace directly from information about the workspace. Then we discuss several possibilities for the potential field that can be constructed on the grid, each of those potential fields expressing different properties of the robot arm motions in the Workspace and leading to different planning strategies. Finally we discuss the problem of adapting the grid and the potential field on it, when changes in the Workspace occured. The construction of the non-uniform grid on Cspace and the construction of the potential field can only be considered as a preprocessing step for the planning problem, when some changes in Workspace can be handled without recomputing completely the grid and the potential field.

We concentrate on a robot with rotational joints only, and to Cspaces of at least 6 dimensions. Recently, we have developed an algorithm for path planning in Cspace, using a particular representation (so called GET-representation) of a non-uniform grid on Cspace and a simple 'potential field' on the grid (see [Van Geem 94a] and [Van Geem 94b]).

The cells of the non-uniform grid, used in our planning algorithm, can be classified in two categories: 'white cells' and 'black cells'. 'White cells' are those cells in which all configurations correspond to free positions in workspace; 'black cells' are cells in which some (or all) configurations are forbidden positions in workspace (physically not feasible for the robot, or colliding with an obstacle). Here we consider the computation of the GET-representation of such a non-uniform grid on Cspace, when given a robot arm and a number of obstacles in the workspace of the robot.

This paper is structured in the following way. In Sections 2 to 5 the construction of the GET-represetation a non-uniform grid on the Cspace of a robot with 6 rotational joints is discussed. In Section 2 the construction of Cspace is analysed and a subproblem, a 2 dimensional intersection problem, is solved. In Section 3 a complete algorithm for the construction of a non-uniform grid on Cspace is described when only one obstacle is in the Workspace of the robot arm. In Section 4 the merge of two GET-representations for the same robot arm but with different obstacles in its workspace is described, which makes it possible to compute the GET-representation of a non-uniform grid on the Cspace for a robot with 6 rotational joints in an environment with multiple obstacles. In Section 5 the algorithm for the construction of the GET-Representation of the non-uniform grid on Cspace is given. In Section 6 we discuss several possibilities for computing a potential field on the non-uniform grid. We discuss the different motion planning strategies corresponding to the different potential fields. And in Section 7 we describe operations on the non-uniform grid and the potential field when in Workspace an obstacle has disappeared, is added, or slightly moved or when the non-uniform grid is refined nearby an obstacle (in order to model the obstacle more accurately).

2 The Computation of a Grid.

We model the robot arm hierarchically, using cylinders for the links and considering an approximation of the volumes swept by a number of links. In order to determine the angles where collision occurs with the obstacle, we will compute the angles for which an intersection occurs for one of teh cylinders and for the swept volumes. We reduce the 3D intersection problem to a number of calls of a Basic Operation that solves a 2D intersection problem. With this Basic Operation we find candidates for the extreme values of the angle by simple computations on the input.

2.1 Modelling Scheme.

The GET-representation of a non-uniform grid is a compact representation of the generalised quadtree (2^k-tree) corresponding to the grid. It consists of two binary lists. Due to the characteristics of generalised quadtrees and their GET-representation, it is sufficient to consider only the problem of constructing the Cspace with one obstacle in the workspace. For the computation of the GET-representation for two obstacles, we can combine the two GET-representations where we considered each of the obstacles seperately. The operations on 2^k-trees needed for this are, joining two trees to a new one and deleting subtrees. These operations are easily translated to simple operations on the lists of the GET-representations. These operations on GET-representations will be given in Section 4.

We use the same modelling scheme for the robot arm as the one described in [Tournassoud & Faverjon 88]. In that article, a hierarchical solid modelling technique is used to model the robot arm. A robot arm can be seen as a chain of links, each modelled by solid bodies, each of them connected to the next link by a joint. At one of the ends of this chain, the robot arm is fixed to the environment. We start counting the links and the joints from this fixed link on. That is, we denote A for the robot arm, consisting of links A_0, A_1,..., A_k, where A_{j-1} and A_j are connected to each other by the j-th joint (for $j = 1, ..., k$) and A_0 is fixed to the environment (e.g. stands on the floor of the workcell).

On the j-th hierarchical level, the links A_0,...,A_{j-1} are modelled by j solid bodies. The other links are modelled by their swept volume with respect to the joints following link A_{j-1}. The cells on the free space in Cspace are computed stepwise. In the j-th step, the j-th dimension of the Cspace is added. The $j-1$-dimensional white cells computed in the previous step are expanded to j-dimensional white cells; additional free space is computed for the $j-1$-dimensional black cells. Indeed, the swept volume considered in the previous step has led to area defined as forbidden for all values of the j-th parameter, while usually for some values of the j-th parameter there is no collision with the obstacle. So, in each step we have to compute, in those areas considered as forbidden in the previous step, the intervals for the j-th parameter for which no collision occurs. In [Tournassoud & Faverjon 88], these intervals are approximated using a heuristic approach for the computation, and regroupment of very small cells is necessary for the construction of the generalised quadtree. We approximate the cells geometrically, immediately looking for the GET-representation, and keeping track of the rate of approximation of the obstacle by the grid. Another difference with the approach in [Tournassoud & Faverjon 88] is, we do not allow the robot to access partially free cells ('grey cells') in our planning phase.

For simplicity reasons, we consider the obstacle in 3D workspace to be a convex polyhedron. We assume that the links of the robot arm are modelled by cylinders. In the j-th step, we are looking for a safe but accurate approximation for particular extreme values of the j-th parameter, for each $j-1$-dimensional black cell. These extreme values will be the endpoints of the intervals along the boundary of the j-dimensional white cells, in the j-th direction. In order to find these extreme values, we have to compute an approximation of the values of the j-th parameter for which an intersection occurs.

2.2 Intersecting a Polyhedron with a Cylinder.

The cylinder rotates around an axis through the center of one of its circular faces. Consider the center of this circle to be the origin of a fixed coordinate frame, and the rotation axis to be the Y-axis. We choose the symmetry axis of the cylinder to be the Z-axis and we choose the X-axis in

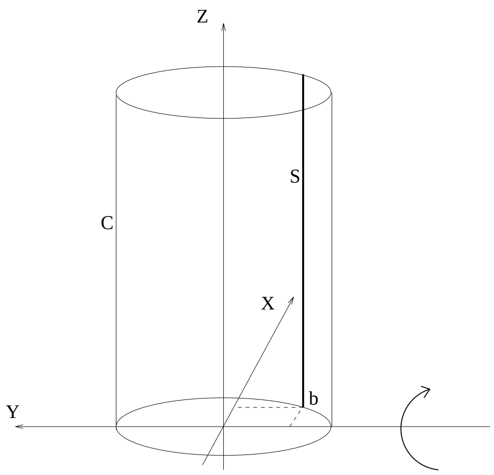

Figure 1: Rotating Cylinder C.

such a way that we get a righthand oriented coordinate frame (see figures 1 and 2).

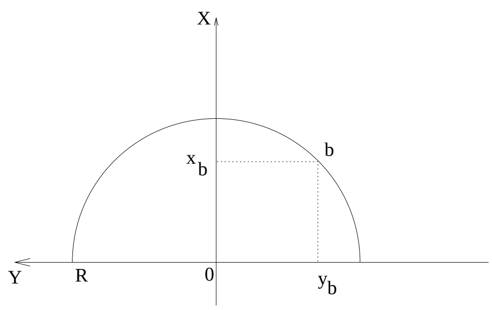

Figure 2: Situation in the XY-plane.

We then consider the interval along the Y-axis that intersects the cylinder. We preform a subdivision strategy along this interval in order to find the extreme values for the angles of rotation for which intersection occurs. For each step in the subdivision, we look for intersections in a plane perpendicular to the Y-axis.

2.2.1 The 2D Intersection Problem.

In each step of the subdivision, we choose some particular y-value, and we only consider what happens in plane Φ parallel to the Z-axis and through the point on the Y-axis with the chosen y-value.

The intersection of plane Φ with the cylinder is a segment S parallel to the Z-axis. (Due to symmetry, we only consider wlog. the halfplane with positive x-coordinates.) We compute the distance in plane Φ of the endpoints of this segment to the origin. For the endpoint b lying in the XY-plane, this is just its x-coordinate x_b. We denote the other value by r_{max}.

When rotating the cylinder, each point p of segment S describes a circle $C(o, r)$ with center in the origin and radius the distance r for p to the origin. All possible values for r are between x_b and r_{max}. Figure 3 shows the situation in plane Φ.

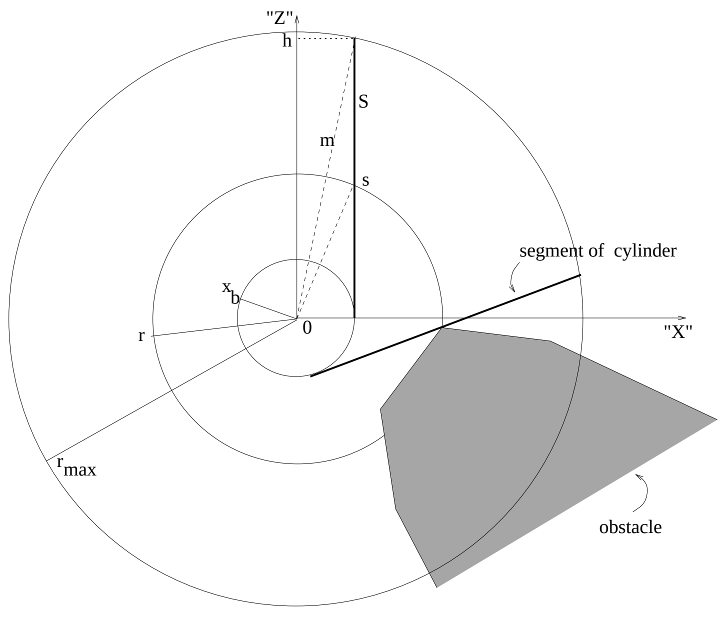

Figure 3: Line L on the cylinder sweeps in plane Φ.

The intersection of plane Φ with the polyhedron can be empty, or some polygon (possibly just one point). If the intersection is non-empty, some interesting points w on the polygon are the vertices lying on a distance from the origin between x_b and r_{max}, and the intersection points of the edges adjacent to only one such a vertex with circle $C(o, r_{max})$ or with circle $C(o, x_b)$. In the following, we denote the finite set of these points w by W.

Observe that for a given point w in the circular region swept by S when rotated, there is exactly one point s on the segment that collides with a given point w during the sweep. Indeed, this point s has coordinates $(x_b, y_b, \sqrt{r^2 - x_b^2})$, where r is the radius of the circle with center in the origin and through w. So, for any points w on the polyhedron, we can easily determine the unique angle α_w for which the segment, rotated over that angle, intersects with the polyhedron at w. This computation is the Basic Operation in our approach.

We could also find the extreme values for the angle in plane Φ. Since W is a finite set, we can easily determine the extreme values among the angles α_w corresponding to points w in W.

2.2.2 Subdivision.

In the following, we will need some notations. Let C denote the cylinder in its initial position (angle of rotation $\alpha = 0$), C_α for the cylinder rotated over angle α, P the polyhedron and e an edge of P. Let us write R for the radius of the circle at the basis of cylinder C. Then we start off with the interval [-R,+R] along the Y-axis.

Note that an intersection of P with an infinite cylinder, when the latter rotates, happens first at an edge of P. For the finite cylinder C, the first intersection might also happen at a face. This observation was already encountered in the previous: in the 2D intersection problem we mentioned the vertices of the slice that is cut out of P by plane Φ (these correspond to edges of P) and the intersection points of this slice with circles $C(o, x_b)$ and $C(o, r_{max})$ (these correspond to faces of P).

In the following, in the case of a first intersection with a face F, we have to consider an artificial edge e being the intersection of F with the plane through the circle at the top or the bottom of C_α (respectively for the cases where the contact was made with $C(o, x_b)$ or $C(o, r_{max})$). We can now make a few observations.

Observation 1 *Suppose that the cylinder intersects polyhedron P in vertex v of P, for angle α, when rotating by increasing the angle. Let e be an edge of P incident to v. Then three cases can occur:*

case 1: e intersects the cylinder 'earlier' than v, i.e. for smaller angle than α *(this is the case in which we are interested most),*

case 2: e intersects the cylinder 'later' than v, i.e. for larger angle than α.

case 3: e does not intersect the cylinder for any angle,

Observation 2 *In the same settings as above, the three cases can be distinguished from eachother. The relative position of edge e w.r.t. cylinder* C_α *must be one of the following three possibilities:*

pos. 1: the line through edge e intersects the cylinder in more than one point but only v is on e,

pos. 2: the edge e intersects the cylinder in at least one point p other than v $(p \neq v)$,

pos. 3: the line through e is a tangent line of C_α *in v.*

Possibility 1 corresponds to case 1, possibility 2 corresponds to case 2, possibility 3 corresponds to case 3.

A similar observation can be made for points anywhere on an edge of the polyhedron:

Observation 3 *Let* Φ_1 *be a plane parallel to the XZ-plane, intersecting cylinder* C *in line* L. *Let* α_1 *be the angle for which line* L, *when rotating from low to high values for the angle, touches the polyhedron for the first time (a result of the computation as described in the previous section). Let* p_1 *be the point of intersection of* L *and edge* e *of the polyhedron. Then we can distinguish two kinds of positions of* e *w.r.t. the rotated cylinder* C_{α_1}:

- $e \cup C_{\alpha_1} = \{p_1\}$: *then* p_1 *is the first intersection of the rotating cylinder with edge* e *and* α_1 *is an extreme value for the angle along edge* e,
- $|e \cup C_{\alpha_1}| > 1$: *then there must be another point* p_2 $(\neq p_1)$ *along edge* e *for which the rotating cylinder intersects edge* e *at angle* α_2, *where* $\alpha_2 < \alpha_1$.

Observation 4 *Let* P *be a polyhedron, let* C *denote a cylinder. Denote* I_P *for the interval of y-coordinates of the points in* P, I_C *for the interval of y-coordinates of the points in* C. *When* P *intersects* C, *we distinguish the following situations and define the 'search region'* SRY *accordingly:*

- *$I_P \subset I_C$, define $SRY = I_P$,*
- *$I_C \subset I_P$, define $SRY = I_C$,*
- *non-empty intersection, but not one of the cases above, define $SRY = I_P \cap I_C$.*

Intersections between the rotating cylinder and polyhedron P can only occur in points of P and C with y-coordinate in 'search region' SRY. The interval SRY is always contained in $[-R,R]$. The y-coordinates of points on an edge or a face of P may all lie in SRY. If this is not the case, then we say that the edge or face is 'clipped by C'.

Observation 5 *When sweeping plane Φ (defined as in the previous section) in the y-direction along interval SRY, the angle for which first contact occurs changes due to two changes:*

- *the intersection of Φ with the cylinder,*
- *the intersection of Φ with polyhedron P.*

These contributions to the change of the extreme value along an edge may annihilate eachother when sweeping along the edge. At a vertex of P, encountered by Φ sweeping by, a dramatic change can be expected. The edge (or artificial edge) where the first contact occurs on will change only when one of its endpoints (a vertex or an artificial vertex of P) is reached by the sweeping plane Φ.

The observations of above lead to the following algorithm.

Step 1: Compute SRY.

Step 2: Subdivide SRY in intervals $[y_i, y_{i+1}]$ such that all (artificial) edges of P with at least one point with y-coordinate in SRY are intersected by at least one plane Φ_i (parallel to the XZ-plane, with y-coordinate one of the y_i's).

Step 3: Compute for each Φ_i the extreme values α_i (where first contact occurs in Φ_i) with the Basic Operation. We now have for each of these α_i's a corresponding edge (or artificial edge) e_i of P.

Step 4: Find for each α_i the endpoints of the interval of extreme values around it, when sweeping a plane Φ along the corresponding e_i. When edge e_i is clipped we only have to consider the part of e_i with y-coordinates in SRY.

Step 5: Find among all these endpoints the smallest and largest value for α.

Since the polyhedron is convex, we have exactly one minimum and maximum angle. In Step 3, we check all edges with some point with y-coordinate in SRY because we do not know a priori which edges are going to give rise to the extreme values of α. In Step 5, we only have to find the minimum and the maximum value in a finite set of numbers. We will concentrate now on Step 4.

2.2.3 Step 4.

We propose two different approaches for the computation in Step 4. The first one is a discretisation approach, the other one is analytic with a difficult symbolic computation problem.

Discrete. We must make sure that the value for α is a safe one, because we want to be sure that the free cells in the non-uniform grid are really empty. I.e. we have to know about the behaviour of the values for α along edge e when sweeping plane Φ along. We can use Observation 3 for this purpose.

Indeed, plane Ψ through edge e and the over α rotated segment S is tangent to the cylinder iff. α is an extreme value along e. Thus, for each (artificial) edge e, we can proceed in the following way.

i: Take arbitrary point w on edge e (say, w is the midpoint of interval $[p_a,p_b]$, where p_a and p_b are the endpoints of e).

ii: Compute α_w for this point w (using the notation of the previous section).

iii: Rotate points p_a and p_b over angle $-\alpha_w$, resulting in points e_1 resp. e_2.

iv: We now check whether the line through e_1 and e_2 is tangent to cylinder C. We can do this check in the XY-plane by projecting the points e_1 and e_2 and determining whether the line through their projections is tangent to the circle at the bottom of the cylinder. So, project e_1 and

e_2 to the XY-plane, compute the square of the distance of the line E through these projected points to the origin and compare the result with R^2 (the radius of cylinder C). This is shown in figure 4. If these values are identical, α_w is the extreme value for angle α along edge e.

v: If these values are different, then the extreme value for α along edge e occurs at another point w on e.

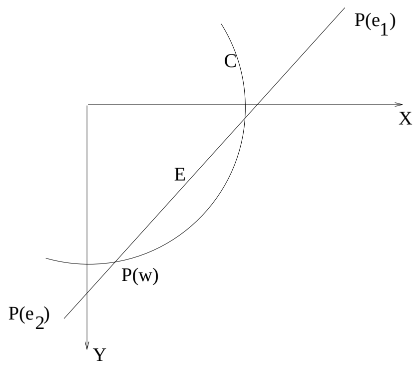

Figure 4: Check for tangency after projection P to the XY plane.

We now concentrate on this last step. If indeed point w, chosen as above, is not the good one, we adapt α_w by a factor $\sigma > 0$. We can consider the angle $\beta_1 = \alpha_w - \sigma$ as a new candidate for the extreme value. We now proceed with a procedure similar to steps [ii] to [v], but with fewer computational effort and without computing the to β_i $(i = 1, 2, ...)$ corresponding point $\hat{w}$. That is, we compute the projection E_i of edge e after it has been rotated over $\beta_i = \beta_{i-1} - \sigma$ (where $\beta_0 = \alpha_w$), and check for the number of intersection points with the circle. If there is one intersection point, then we reached the extreme value for the angle. If there are two intersection point, then we repeat the adaptation of α_w (we consider $\beta_{i+1} = \beta_i - \sigma$). If there is no intersection, we repeat the adaptation of α_w in the opposite direction by $\frac{1}{2}\sigma$ (we consider $\beta_{i+1} = \beta_i + \frac{1}{2}\sigma$). Obviously, we again use the technique of computing the distance from the origin to E_i, in order to determine which of the above cases apply. We should keep in mind that here we compute numerically the distance and the comparison with radius R, so some inaccuracy in the data may occur.

In order to compute E_i, we can use a formula. Rotating p_a and p_b over angle κ and then projecting the resulting points on the XY-plane, we get two dimensional coordinates

$$(x_{pa} \cos \kappa - z_{pa} \sin \kappa, y_{pa}) \text{ and}$$

$$(x_{pb} \cos \kappa - z_{pb} \sin \kappa, y_{pb})$$

respectively. So, the slope s_i of E_i in the XY-plane is then:

$$s_i = \frac{y_{pb} - y_{pa}}{(x_{pb} - x_{pa}) \cos \kappa - (z_{pb} - z_{pa}) \sin \kappa}.$$

And thus, the equation for E_i is then:

$$(y - y_{pa}) = s_i \ (x - (x_{pa} \cos \kappa - z_{pa} \sin \kappa)).$$

Having this equation, we can now compute the distance of the origin $(0,0)$ to E_i:

$$d((0,0), E_i) = \frac{y_{pa} - s_i \ (x_{pa} \cos \kappa - z_{pa} \sin \kappa)}{\sqrt{(y_{pb} - y_{pa})^2 + ((x_{pb} - x_{pa}) \cos \kappa - (z_{pb} - z_{pa}) \sin \kappa)^2}}.$$

The value of σ will be chosen appropriately to get an adequate subdivision for a 2^k representation of the Cspace and thus suited for the GET-representation.

Symbolic. The problem of finding the appropriate α can be reduced to the problem of finding a minimum of a function f. The angle α can be written as a function of some parameters. An extreme value for α is then also an extreme value for that function f. In the following, such a function f is constructed. However, we did not find a suitable way for the determination of the extreme points of f.

Given the coordinates of the two (artificial) endpoints of edge e, and an arbitrary point p on e in between these two endpoints, we can compute the equation of plane Φ through p and parallel to the XZ-plane. We parameterize p in function of the two endpoints p_a and p_b:

$$p = \lambda p_a + \mu p_b,$$

where $\lambda, \mu \in [0,1]$. For this plane Φ we can compute the extreme values for angle α. Doing so, we get a function f in two variables λ and μ. The minimum and the maximum value of f for $\lambda, \mu \in [0,1]$ are the extreme values of α along edge e. This can be done as a preprocessing step.

However, f is a highly non-linear function. Indeed, when we write (x_{pa}, y_{pa}, z_{pa}) and (x_{pb}, y_{pb}, z_{pb}) for the coordinates of points p_a and p_b respectively, then the equation of plane Φ through point p and parallel to the XZ-plane is

$$\Phi : y = \lambda y_{pa} + \mu y_{pb}.$$

Line L being the intersection of Φ with C gives us an equation for x_b:

$$x_b^2 + (\lambda y_{pa} + \mu y_{pb})^2 = R^2.$$

Let r be the radius of the circle in Φ as in the algorithm of the previous section, on which the contact point p lies. That is,

$$r = d(o_\Phi, p), \text{ where } o_\Phi = (0, \lambda y_{pa} + \mu y_{pb}, 0).$$

Observe that $r \in [x_b, r_{max}]$. Let $\hat{r} = \sqrt{r^2 - x_b^2}$, and point $c = (x_b, \lambda y_{pa} + \mu y_{pb}, \hat{r})$. We know that point p is the point reached by point c after rotation by angle α. The angle α is the angle between the lines $o_\Phi p$ and $o_\Phi c$ in plane Φ.

We now express α as a function in λ and μ. Let γ be the angle such that

$$\tan\gamma = \frac{\lambda z_{pa} + \mu z_{pb}}{\lambda x_{pa} + \mu x_{pb}}.$$

(The angle γ can be seen as the angle of rotation from one coordinate frame to another needed for a straightforward computation of $\tan\alpha$.) Then

$$\begin{aligned} \tan\alpha &= \frac{x_b \sin\gamma - \hat{r}\cos\gamma}{x_b\cos\gamma + \hat{r}\sin\gamma} \\ &= \frac{x_b\tan\gamma - \hat{r}}{x_b + \hat{r}\tan\gamma}. \end{aligned}$$

This gives us the function f. (In fact, we can use $\tan\alpha$ instead of α as the parameter for the angle of rotation of the joint of the robot, when we assume that the rotational joint cannot rotate freely over 2π.)

2.3 Intersection of a Polyhedron with a Swept Volume.

We propose to use a similar approach as in the previous for the computation of the 'extreme angles' for swept volumes. However, the swept volume is a more complicated object considered as geometric model. Therefore the approach of above will lead to some more complicated computations.

In a first attempt to find a safe bound for the 'extreme angles' we can model the swept volumes by cylinders that contain the swept volumes completely.

3 The Hierarchical Algorithm for One Obstacle.

Compute for Cylinder A_1 and angle α_1 the black cells where A_1 intersects the obstacle (polyhedron) P. Then, for each white cell already computed, find the ones where the swept volume SV_1 for the rest of the robot intersects P. These cells are (partially) forbidden area, and therefore we can call them 'grey cells'. Here a further split-up of the grey cells can be expected. On the white cells in the first dimension of the Cspace (corresponding with α_1) where SV_1 does not intersect P, we can build a white, 2 dimensional area. Subdivide those grey cells where SV_1 does intersect P in intervals small enough for a good approximation of the free space. For each value of α_1 in this subdivision, consider cylinder A_2 and α_2.

For each value of α_1 to be considered, compute the intervals for α_2 for which there is an intersection of A_2 with P. This results in a 2 dimensional grid. In those white cells of the 2 dimensional grid where SV_1 did intersect P (the grey cells, as we called them), we now compute where the swept volume SV_2 intersects P. Here again a further split up of these grey cells can be expected. Subdivide the cells where SV_2 intersects P in intervals for α_2 small enough for a good approximation of the free space. We then proceed as before until all links of A are treated.

3.1 Some Comments.

Observe that the accuracy of the approximation of swept volumes only influences the amount of computation to be performed. A better approximation of swept volumes than by cylinders makes the intersection problem harder to solve. Therefore we limit ourselves to the approximation of swept volumes by cylinders.

The subdivision of the i-dimensional white cells in order to compute the extreme values for the $i+1$-th dimension leads to a discretisation of the computation. This computation might grow exponentially. However, for each new dimension added to Cspace, the number of cells that have to be considered will be reduced.

The subdivision of the i-dimensional white cells can be done in an appropriate way w.r.t. the GET-Rep. When joining cells together along the $i+1$-th dimension, this can be done appropriately too using the GET-Rep.

3.2 Parallelisation.

By computing the GET-representations for each obstacle seperately and combining these afterwards, we can parellelize the computation of the Cspace (several processors can compute the GET-representation for different obstacles, combining the GET-lists can be done with a parallel cascading technique). In this way, we estimate that the computation of the representation of the grid on Cspace can be done fast enough in order to compute it for a Cspace of at least 6 dimensions.

4 Merging the GET-Representations.

In this section, we consider the merging of two GET-representations, each corresponding to the same robot but with another set of obstacles considered in the workspace. The GET-representation consists of two lists with binary entries. When a non-uniform grid over a k dimensional Cspace is represented by a 2^k-tree, the GET-representation is a very compact description of the tree. It describes the structure of a 2^k-tree with the first list (DFA). With the second list (DFB) it describes whether the cell in the corresponding grid is accessible (white) or not (black). In order to construct these two lists, we traverse the 2^k-tree in breath first order. For all nodes (except for the root) we write as entry in the DFA list a '0' when the node is a leaf and a '1' when the node is not a leaf. For each leaf in the tree, we also add a '1' as entry to the DFB list when the corresponding cell is black, and a '0' as entry to the DFB list when the corresponding cell is white. More details on the GET-representation can be found in [Burger 92] and [Van Geem 94b].

In order to explain clearly our method, we will consider in the first subsection the DFA-lists only. In the second subsection, we will present the difficulties occuring when only the DFA-lists are considered and in the last subsection we present the complete algorithm for the computation of the merge of the two GET-representations.

4.1 Merging DFA's, Special Case.

For simplicity, we first consider an example in the two-dimensional case. The generalisation, also to higher dimensions, is straightforward.

Consider the following example. Given are the DFA-lists:

1010 00000100 0000,

1001 00010100 00000000.

The spaces in the lists indicate the levels in the corresponding quadtrees. The partitioning is illustrated in figure 5.

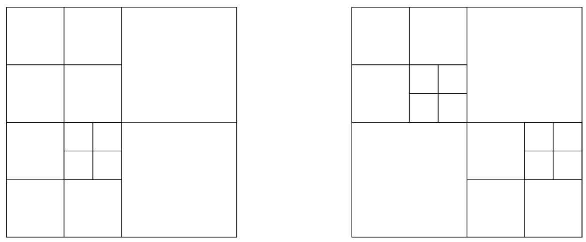

Figure 5: Partitionings corresponding to the DFA-lists.

We now take the 4-tuples in the first level:

1010,

1001.

The i-th entry of the first 4-tuple of the merged DFA-list will be the logical OR of the i-th entries of the original 4-tuples. This results for the example considered in:

1011.

For the second level, we first add some dummy 4-tuples. A 4-tuple in the second level corresponds to each 1 in the first level. We add a dummy 4-tuple in the second level for each 0 occuring in the first level, on the spot where we expect a 4-tuple when there would have been a 1 on the position of the corresponding 0 in the first level. We denote a dummy 4-tuple by '$$$$'. In the example we get for the second level:

0000$$$$0100$$$$,

0001$$$$$$$$0100.

We now apply OR again as in the previous, where $ OR $ is defined as $ and $ OR 1 = 1, $ OR 0 = 0, 1 OR $ = 1 and $ OR 0 = 0. This results in:

0001$$$$01000100.

We delete the $ from the resulting list and repeat the process described above for all following levels. The resulting merge in the example considered is:

1011 000101000100 000000000000.

This corresponds to the partitioning illustrated in figure 6.

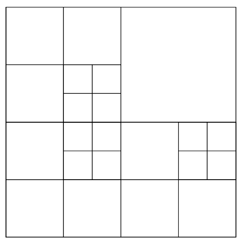

Figure 6: Result of joining the DFA-lists.

For the k dimensional case, the method described above can be generalised. The only difference is that we now have to manipulate tuples of length 2^k.

4.2 The Cases For Which Things Are More Complicated.

Things become slightly more complicated when we also consider the DFB-lists. Indeed, considering the example of the previous subsection, with the following DFB-lists respectively:

0000000001011,

$$0100000000011010,$$

the resulting DFA-list is not the expected merge:

$$1001\ 00010100\ 00000000 \text{ (the expected DFA-list);}$$

in the merge of the previous subsection there are too many small-sized cells. This is illustrated by figure 7 corresponding to the original DFB-lists, and by figure 8 corresponding to the expected DFA-list after merging (which the reader might want to compare with figure 6 corresponding to the result of the procedure described in the previous subsection).

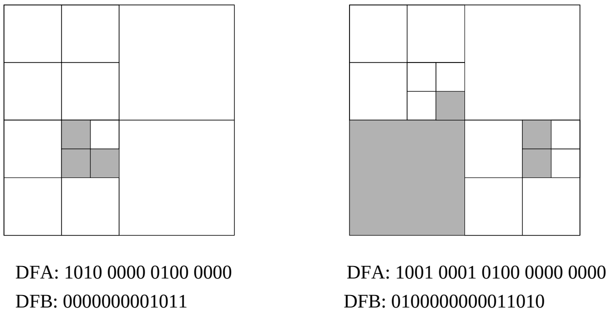

Figure 7: Partitioning corresponding to the GET-representations.

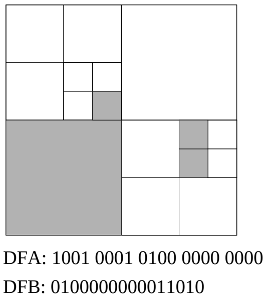

Figure 8: Expected Result.

The way to handle this observation is, by checking on the entries of the DFB-lists while merging the DFA-lists. This, as well as the construction of the DFB-list corresponding to the DFA-list computed by the merge, will be described in the following subsection.

4.3 The Algorithm For Merging In All Cases.

For the computation of the merge of two DFA-lists, we adapt the algorithm given in the first subsection only slightly, by the introduction of a test on the entries of the DFB-list. We will now introduce also the merge of the DFB-lists.

The Cspace is now assumed to be k-dimensional, $k > 0$.

We consider a function j to be given, computing for entry i in a DFA-list, the corresponding entry in the DFB-list.

When a large cell occupied by an obstacle is discovered in one of the DFA-lists, all children in the 2^k-tree corresponding to the other DFA-list have to be removed. We replace them by dummy 2^k-tuples, consisting of $-entries only.

IN: (DFA1,DFB1),(DFA2,DFB2),

OUT: merge (DFAr,DFBr).

- introduce dummy 2^k-tuples
- for (i=0; i < max(length(DFA1),length(DFA2)); i++)
  - if (((DFA1(i)==0)&&(DFB1(j(i))==1)) ||
    ((DFA2(i)==0)&&(DFB2(j(i))==1)))
    - DFAr(i)=0;
    - DFBr(j(i))=1;
    - adapt next levels: replace in DFA2 or DFA1 resp. the corresponding 2^k-tuple by $...$ (2^k times).
  - else
    - DFAr(i)=(DFA1(i) || DFA2(i));
    - if (DFAr(i)==0)
      - DFBr(j(i))=(DFB1(j(i)) || DFB2(j(i)));
- return (DFAr,DFBr);

5 Hierarchical Construction of GET-Rep.

In order to construct the GET-Rep in a hierarchical way, we need two routines. The first one adds an extra dimension to the Cspace, the second one processes the information from the computation of new black and white cells in the cells known to be grey in the previous hierarchical step.

Suppose we start with the GET-Rep of an i-dimensional Cspace. Since we want to keep the cell construction symmetric (we want the cells to be $i+1$ dimensional cubes), we first have to 'double' the GET Rep on each level of the corresponding 2^i-tree. Since we have to 'double' all children of a 'doubled' '1' entry in the DFA list of the GET-Rep, we best 'double' the GET Rep level by level. For example, starting from a 1 dimensional Cspace with DFA

10, 00

we double the first level to

1010, 00 00

and then the second level to

1010, 0000 0000

. 'Doubling' a level, consist of two actions. First we transform a 2^i tuple T into a 2^{i+1} tuple by concatenating two copies of the original 2^i tuple. And second, we insert on each level a copy of the 2^i tuples corresponding to the subtrees of the '1' entries in tuple T on the appropriate place. When this is done for all 2^i tuples on level j, we repeat the process for all 2^i tuples on level $j+1$. This is the first of the two routines. We illustrate the routine once more with an example of a GET Rep of a 2 dimensional grid to a 3 dimensional one. Say we start with

1010, 0000 1000, 0000.

Then doubling on the first level results in

10101010, 0000 1000 0000 1000, 0000 0000.

Doubling on the second level results in

10101010, 00000000 10001000 00000000 10001000, 0000 0000 0000 0000.

Finally we also double the third level, resulting in

10101010, 00000000 10001000 00000000 10001000, 00000000 00000000 00000000 00000000.

This process has to be carried out analogously for the DFB list of the GET Rep.

The second operation consists of splitting up some $i+1$ dimensional cells in smaller ones. Here we have to turn some '0' entries in the DFA list into '1' entries, corresponding to the extreme values for the $i+1$-th parameter computed during the hierarchical construction of the $i+1$-th dimension of Cspace. In the DFB list some cells will be marked white and others black.

6 Construction of Potential Function.

6.1 The Non-hierarchical Way.

In this approach the GET-Rep for the whole Cspace of the robot is given as an input to the assignment of values for U to the cells of the non-uniform grid. A non-hierarchical computation of U is described in [Van Geem 94b]. This approach is more suitable for planning a path when it is expected that moves of all joints have to be considered in order to find a path. This might be the case when the robot arm carries a large object at its end effector, since then the orientation of that large object is relevant and since this orientation usually is determined by the last joints.

6.2 The Hierarchical Way.

The main motivation for thinking of a hierarchical way of constructing U is that then it might happen that we can already find a path in a low dimensional Cspace. Since Cspace is constructed starting from the joint closest to the fixed link of the robot arm, executing a path found in a low dimensional Cspace will result in moves of the first links only. These links are usually the ones that are moved preferably. This planning approach leeds in a natural way to gross motion planning, by specifying the goal position of the robot configuration as a large cell. Additional fine motion could then be planned locally, in and about the large cell that was specified as the goal in the gross motion planning faze.

In a grid there are three kinds of cells: white, grey and black cells. If we would give a U value to grey cells, a path could be planned through a grey cell. This can cause the well known problems of planning with grey cells and heuristic approaches, since we are not sure that a path through a grey cell is going to be feasible: this depends on the links that were not yet considered for the construction of the Cspace. Theoretically it can even happen that a grey cell turns black completely. Therefore we decided not to give a U value to the grey cells.

We can start computing a potential field as soon as the initial configuration of the robot is in a white cell. We then can give a U value to each white cell that can be connected with the cell where the initial configuration is by a path through white cells. Some white cells may not be reached. They might lie in another connectivity component of the free space in Cspace or currently unreachable because some grey cell are lying in the way.

When a new dimension is added to the Cspace and grey cells turned (partially) into white, grey or black (only grey cells turn into grey ones), we can determine in which cell (for the new dimension) the initial configurations lies. Then we restart the wave from that cell, forgetting about the potential field computed in lower dimensions.

6.3 The Increments and Path Planning.

The 'filling in' of the cells of the non-uniform grid with U values by the wavefront approach consists mainly of the inheritance principle. A cell c gets a value if there is a neighbouring cell that already has a value. The value at c is the value at the neighbouring cell, incremented by a fixed value (the increment). We can choose the same increment for all inheritance, as in [Van Geem 94b]. But we can also consider different increments according to the situation. We consider three strategies.

1: We use a different increment for each joint. This means that the increment is different in the different directions in Cspace.

2: We use a different increment when we inherit from a large cell to a smaller one, than when we inherit from a cell of equal size or when we inherit from a cell of smaller size.

3: We combine the previous two strategies, where we use the increment value of the first strategy as a multiplicative factor for the increment value of the second strategy.

We will now give an example of the three strategies on a 2 dimensional Cspace. In strategy 1, the increment value for the first link is set to 1 and, for the second link it is set to 2. In strategy 2, we use 2 as increment for an inheritance from a cell of equal size, 1 for an inheritance from a small cell to a larger one, 4 for an inheritance from a cell that is one level higher

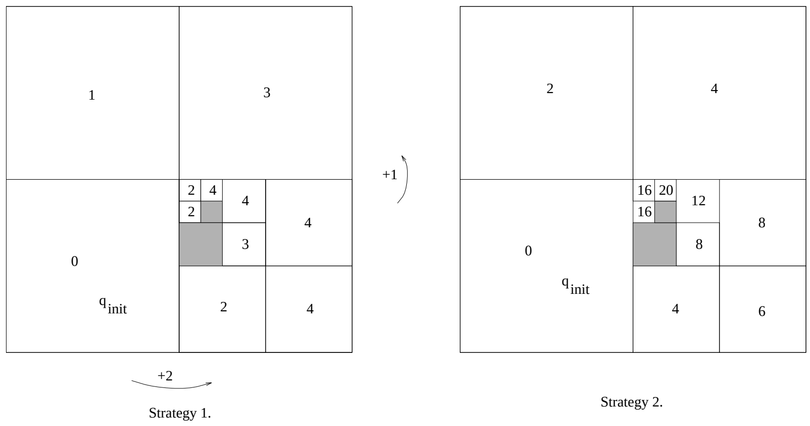

Figure 9: Strategies 1 and 2.

in the quadtree, 8 for an inheritance from a cell that is two levels higher in the quadtree, and 16 for an inheritance from a cell that is three levels higher in the quadtree. The wave starts from the cell containing the initial configuration.

Computing the path after placing the appropriate U on the grid is done by moves from a cell to the neighbouring cell with lowest value for U. This means that all neighbours have to be checked on their U value and the cell with the lowest value has to be found. When this planning strategy is used, the three strategies lead to different meanings of the concept of a 'good' path. In Strategy 1 some directions are prefered more than other ones. Strategy 2 supports the preference to move towards larger cells. Both elements of the two other strategies are present in Strategy 3.

The non-uniform grid and the common property of all strategies of above to move preferably towards larger cells, model the 'repulsive force working on the robot', pushing the robot away from the obstacles.

No local extrema will be produced by any of these strategies because there is only one wave whereby the U value constructed is larger than the value in the neighbouring cell from which the U value is inherited.

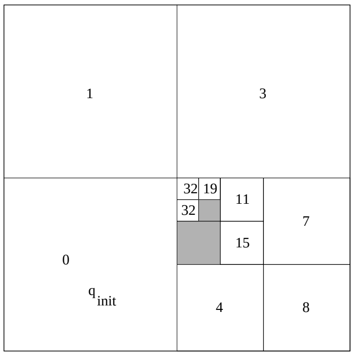

Figure 10: Strategy 3.

7 Operations on Constructed Cspace and U.

The operations discussed in this section appear whenever some change in the environment of the robot is observed. We start with a given model of the Cspace, in the form of a GET-Rep and a potential field U on the non-uniform grid. Overall we can say that there are two kinds of updates for U. The first one involves only local changes to the grid and to U. The second one involves the impact of the change on long distance from the place where the event took place. When we only apply the first one, we still get a potential field without discontinuities and with the certainty that a path will be found if there exists one. However, we loose the nice property that the path will be a pay-off between safe and short path. The second kind of update of U will repair this property by recomputing over a long distance the values of the potential function. We will call the 'wavefront' computation for the first kind of update the *local wave* and for the second kind of update the *update wave*. We now come to the four operations.

7.1 Delete an Obstacle.

When an obstacle has to be deleted, some black cells turn into white ones. These new white cells form a new white area. Sometimes deleting an obstacle also implies that 2^k cells can be merged to one larger cell, and this can happen subsequenty on several levels. Locally the potential field can be 'repaired', in the sense that we can assign a U value to the new white cells just by looking at its direct neighbours. This is just a local wave.

The local wave works in the following way. Find among all neighbours of the area of new white cells, the one with the lowest value and start up a wave in the new white cells from there. The values assigned to the new white cells have to lie between the lowest and the highest U value among the neighbouring cells of the new white area. The increment has to be small enough in order to make sure that the U values for the new white cells do not go over the higher bound. This increment can be determined by considering the amount of new white cells and their size. If the whole new white area corresponds to exactly one new cell (after merging together several smaller cells), the U value from the local wave will simply be the minimum value of all values on the merged smaller sized cells.

To perform the update wave, we start from the cell(s) in the new white area with a neighbour with lowest U value of all neighbours of the new white region and proceed in the usual way (described in [Van Geem 94b]). The update wave does not update those cells (outside the new white area) that have already a U value equal or smaller than the value the wave would assign to them. The wave does not go on from such cells.

It is irrelevant of how many connected components the new white area consists since the update wave disperses first from all cells with the current U value and only after that step from the cells with lowest U value larger than the current one. In the next figure we show such stepwize dispersion of an update wave.

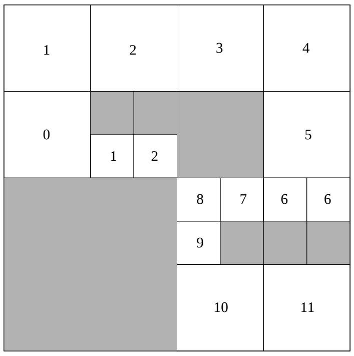

Figure 11: Cspace before deletion.

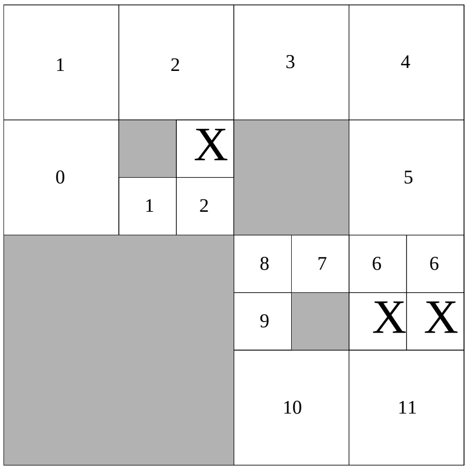

Figure 12: New white cells: X.

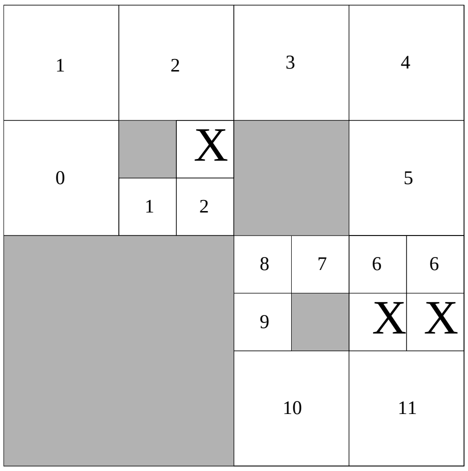

Figure 13: White cells merged.

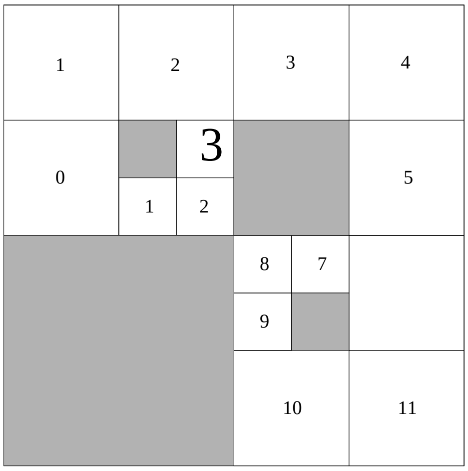

Figure 14: Update wave starts.

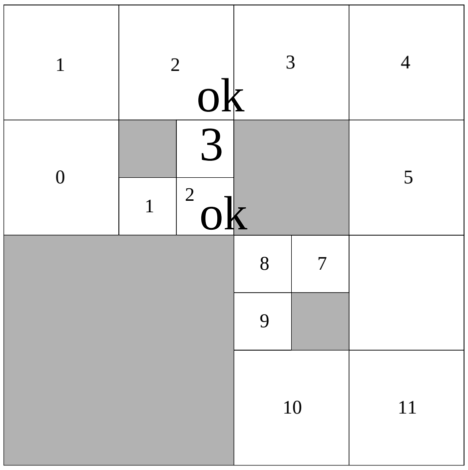

Figure 15: Next step of update wave.

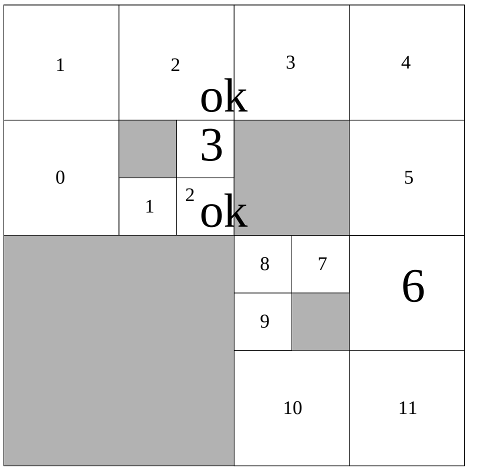

Figure 16: Next step of update wave.

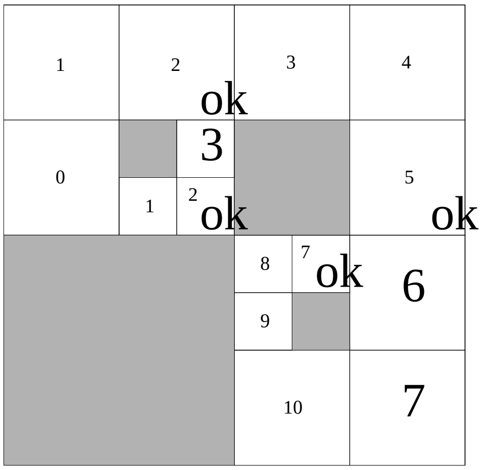

Figure 17: Next step of update wave.

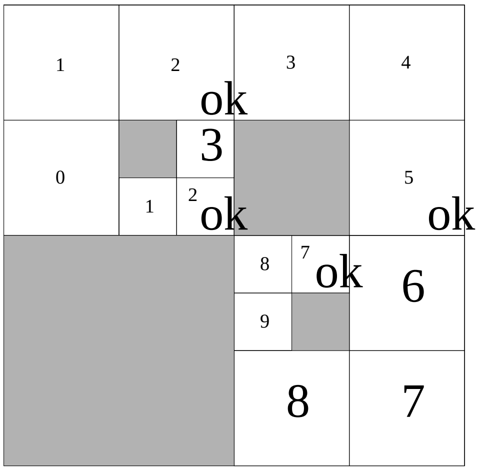

Figure 18: Next step of update wave.

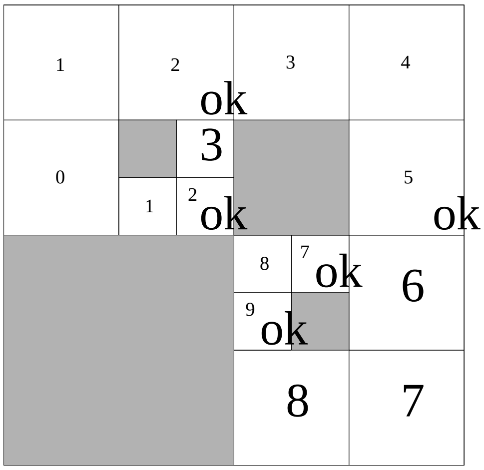

Figure 19: Next step of update wave.

7.2 Insert an Obstacle.

When a new obstacle has been observed, its insertion into Cspace will result in new black cells. These black cells might replace white cells completely or partially. In the latter case, the white cell has to be split up and part of the subcells will become black.

In some cases the new black cells will block a passage in Cspace, in other cases no such dramatic change will occur. These two possibilities can be treated seperately, considering the local wave. When a passage is blocked, we can expect that the local wave must adapt U drastically. When no passage is blocked, a very easy and very local change will be sufficient for fixing up the potential field in order to find 'a' path.

7.2.1 Fixing: No Blocked Passage.

Deciding on whether or not we are in this case can be done simply by looking at the neighbours of the new black area. If they all can inherit their U value from one of their neighbouring cells that were unaffected by the insertion of the obstacle, then we are in this case. This is the first step of the treatment for the local wave anyway, as described in the next subsection.

Intuitively, in this case there is still a passage through the obstacles, 'passing by closely' to the new obstacle. Some previously white cells turned into black ones. These are of no concern to us. Some cells stayed white, and are not affected at all. The only cells to worry about are the previously white cells that are subdivided and partially turned black.

We treat each of these cells as if they were the only cell that was subdivided by the insertion of the obstacle. We look for the largest and the smallest U value of all previously white (and partially or completely white still) neighbouring cells of the previously white cell under treatment. All subcells of the previously white cell under treatment will get a U value between these largest and smallest U values. A wave is started up from the cell(s) where this smallest U value was (were) found. Only the new white subcells get a U value. The increment used in this local wave depends on the size of the subcells (how high the new subtree in the corresponding 2^k tree is), and the strategy used for the construction of the original potential field U. This can be computed with the information available.

7.2.2 Local Wave.

Some neighbours of the new black area inherited their U value from one of the white cells that turned (partially) black and cannot inherit the same value from any of their other neighbours. These cells have to be treated with the local wave we describe in this subsection.

First we search for all the neighbours of the new black area that have to be treated with the local wave. Then we start looking for a cell from where these cells can inherit a new value. Sometimes such a cell does not exist. In that case we have an area in Cspace that cannot be reached from the initial position of the robot (we have another connected component, created by the inserted obstacle). In order to decide whether such a source cell exists and, if there exists one, to find sufficiently many of them, we start up a *wave in the opposite direction* (away from the new black area). We start from the neighbouring cells of the new black area with lowest U values. We call this process also a 'wave' because we have to consider all the neighbours of these cells first before we move to the cells with larger U values neighbouring to the new black area or previously treated cells. We only treat cells that cannot get there U value from any other cell than from one of those treated before or from a white cell now (partially) turned into a black one. For this reason, the wave has to stop. When it stopped, we start up a wave in the usual way (as described in [Van Geem 94b]) from those cells where the 'wave in the opposite direction' died out.

7.2.3 Update Wave.

With the update wave we restore the potential field in the state it should have when U would be recomputed starting from the cell with the initial configuration in it. The computation is very similar to both the local wave and the U computation.

We first decide on which of the unchanged cells are going to be sources for the adaptation of U. We do this by 'a wave in the opposite direction'. The cells visited by this first wave, that have a neighbour whose U value is unaffected by the insertion of the new obstacle(s), are potential sources. Starting from the potential sources, U is adapted with a wavefront approach. A potential source is activated only when the inheritance of the U values reach the value of U at that potential source. We start with this second wave from the potential source(s) with lowest U value among the U values at potential sources.

7.3 Model More Accurately.

Starting from an obstacle and a construction algorithm for Cspace, we get a more accurate subdivision of some black cells. That is, some black cells are subdivided into smaller cells and some of those smaller cells become white. We can assume that no complete black cell turns into a white cell, since we assume that the information we had in the previous grid was correct (i.e. in the black cells some configurations are forbidden, but maybe not all of them). The adjustment of U can now be done exactly like in the approach for deleting an obstacle, described above.

7.4 Move an Obstacle Slightly.

Assume that a non-uniform grid on the Cspace of a robot arm in an environment with several obstacles is given and, that it is observed that an obstacle is not exactly on the position for which the grid was computed. This slight move of this obstacle results in minor changes in Cspace: some larger white cells might be subdivided and turn partially black, some black cells might turn partially white and sometimes get merged with some neighbouring white cells to result in a larger white cell. The adjustment of U can be done in a similar way as for the previously discussed operations, by consecutively adding and deleting black cells.

References

[Burger 92] K. C. Burger, A Kernel-Based Construction System (KBC), diploma thesis, RISC-Linz, July 1992.

[Tournassoud & Faverjon 88] B. Faverjon and P. Tournassoud, Motion Planning for Manipulators in Complex Environments, in 'Lecture Notes in Computer Science, No. 391, Geometry and Robotics'.

[Van Geem 94a] C. Van Geem, Towards a Fast Solution Method for the General Robot Motion Planning Problem using a Manhattan-like Distance Function on a Non-Uniform Grid in Configuration Space, 10th European Workshop on Computational Geometry, Santander, March 17-18, 1994.

[Van Geem 94b] C. Van Geem, On Using a Manhattan Distance-Like Function for Robot Motion Planning on a Non-Uniform Grid in Configuration Space, Technical Report, RISC-Linz, 94-74, October 1994.