A Multiresolution A^* Method for Robot Path Planning

Antti Autere, Johannes Lehtinen
Department of Computer Science, Helsinki University of Technology, Otakaari 1 M, SF-02150 Espoo, Finland
EMail: aau@cs.hut.fi, jle@cs.hut.fi

Abstract

In this paper, a point-to-point robot path planning problem is studied. It occurs in industry for example in spot welding, riveting, and pick and place tasks.

A new A^* -based method is presented. The algorithm searches the robot's configuration space with many different resolutions at the same time. When a path candidate goes far from the obstacles, coarser resolutions corresponding to bigger step sizes is used. When it goes near the obstacle surfaces, finer resolutions corresponding to smaller step sizes is used.

The algorithm always finds a path from a starting robot configuration to the goal configuration if one exists, which is a property of the A^* search in general. This is true given the finest resolution of the search space. These kind of path planning algorithms are called *resolution complete* in the robotics literature.

A^* is also applied because of the possibility to generate better guiding heuristics. A better admissible heuristic roughly means that A^* using it expands fewer configuration space nodes, which is known *a priori.*

A known AI-method utilizing *relaxed models* is applied to generate admissible heuristics. Constructing relaxed models involves removing details from the base level problem to get simplified ones. The heuristics are then obtained by solving these simplified problems.

A simulated robot workcell is provided for demonstrations. The path planning of a 5-degrees-of-freedom industrial robot appears to be reasonably fast.

Keywords: Robotics, Heuristic Searching, A^*, Planning, Path Planning.

1 Introduction

Robot path planning in known environments refers to finding a collision-free path from an initial robot configuration to a desired goal configuration. This problem occurs in industry in spot welding, riveting, and pick and place tasks, to mention some examples. Figure 1 illustrates an example path for a 2-joint robot manipulator. In Fig. 1, "S" is a start configuration and "E" is the goal.

When an automated production line is operating, it is often very expensive to stop it to re-program robots, e.g. to deal with new products. A more economical way is to generate the movements of the robots off-line, in a simulator, and then download the programs into the robot controllers. Minor modifications may still be required but the time the production line needs to be "off duty" is small compared to the manual teaching of the robots.

Path planning has received much attention over the years and many different approaches have been presented. They can be roughly categorized into *potential field*, *cell decompositions* and *roadmap* or *skeleton* methods [14] and [11].

Potential field methods generally employ repulsive potential fields around obstacles and an attractive field around the goal. Path planning is then done by following the negative gradient of the combined potentials. The major drawback is that the potential function will often lead the path to some local minimum, from which it cannot escape without additional help. Many techniques for escaping from local minima have been presented, e.g. in [14] and [3]. It is possible to construct potential functions that are free from local minima but they must be calculated numerically, see e.g. [14], [3] and [5]. However, present computers cannot keep, for example, a 6-dimensional numerical potential function in memory.

Cell decomposition approaches are based on decomposing (either exactly or approximately) the set of free configurations into simple non-overlapping regions called cells, e.g. [8] and [14]. The adjacency of these cells is then represented in a connectivity graph that is searched for a path. The drawback is that all the cells must be constructed before a path can be found. However, recently a hierarchical method to do this has been reported in [1] and [2].

Roadmap or skeleton approaches attempt to retract or map the set of feasible motions onto a network of one-dimensional lines, called the roadmap, skeleton, visibility graph or subgoal network. The path planner tries to connect the start and goal configurations to this network and then search it to find the (optimum) path. The roadmap is usually generated by preprocessing (sampling) the robot's configuration space. There are many ways to do this. Examples of this approach are found in [4], [14], [13] and [12] to mention a few.

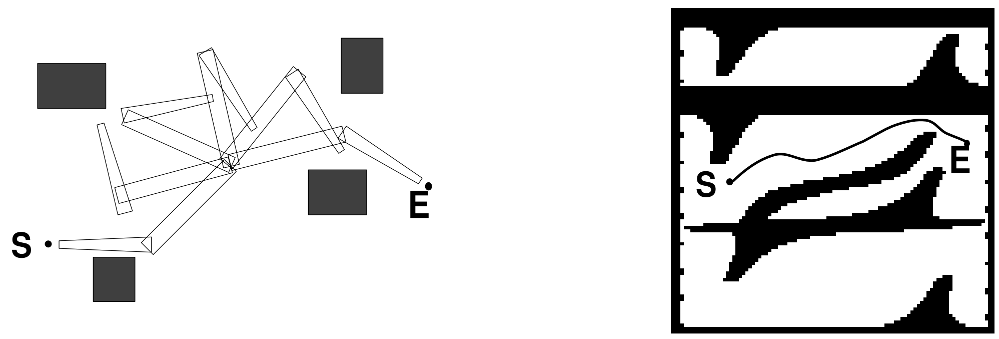

Figure 1: A 2-D path planning example

The nodes of the roadmap or the subgoals can also be constructed by the A^* algorithm. Reference [19] is an example of this. In the reference, A^* first searches with a coarse resolution to produce a set of collision free nodes. Then, it tests whether a collision free path exists from the start to the goal going through some of these nodes. If not, then A^* uses a finer resolution etc.

In the roadmap methods, including the last one, it is usually not known that a path exists from the start to the roadmap nodes or the subgoals. It will be searched afterwards. This may cause unnecessary overhead.

In this paper, we discuss the principles of an A^* -based method that searches with both coarser and finer resolutions *at the same time.* The coarser resolutions correspond to bigger step sizes and the finer resolutions smaller ones. Bigger step sizes are used when the search proceeds far away from obstacles and smaller ones when a path candidate is near the obstacle surfaces. Further, a path from the start configuration to a node *has already been found* before the node is included in the list of subgoals.

2 Robot's Work and Configuration Space

This section briefly introduces some commonly used notions related to robot path planning.

Let A denote the robot, W its *work space*, and C its *configuration* space. The left picture of Figure 1 shows a 2-joint robot manipulator in its work space, and the right picture of Fig. 1 shows the corresponding configuration space. A point $q \in C$ specifies the position of every point in A with respect to a coordinate system attached to W [7]. This point, q, is called a *configuration* of A. All the positions of the points of A specified by $q \in C$ belong to the robot's work space W.

The subset of C consisting of all the configurations where the robot, i.e. its every point, has no contact or does not intersect the obstacles in W is called the *free space* and denoted C_{free}. The complement of C_{free} consists of obstacles and is denoted C_{obst}. C_{obst} is composed of some distinct subsets. These are called *configuration space obstacles* or, for short, C -obstacles. To determine whether a configuration q is in C_{free} or in C_{obst}, in a robot simulator, some geometric calculations are needed. This is called *collision testing.*

Let d be the dimension of C, i.e., the number of the joints or the degrees-of-freedom of A. The configuration $q \in C$ is represented as a d-dimensional vector $(q_1, q_2, ..., q_d)$. Every q_i has m_i distinct values, so C is a graph of $m_1 * m_2 * ... * m_d$ vectors. Let us call it a *configuration graph.*

The graph has edges that connect each node q with all its neighbors. For example, a neighbor of $(q_1, q_2, ..., q_d)$ is $(q_1, q_2 + \frac{q_{max}}{m_2}, ..., q_d)$ with only one component allowed to be changed. So, if C is d-dimensional, then every node has $2 * d$ neighbors.

All searching for path planning is done in the robot's configuration space. We map every configuration q onto a corresponding search node n. The nodes are vectors $n \in Z^d$, where Z^d is the set of vectors whose components are integers, and d is the dimension of C. For example, the neighboring configurations $q = (q_1, q_2, ..., q_d)$ and $q' = (q_1, q_2 + \frac{q_{max}}{m_2}, ..., q_d)$ are mapped onto $n = (n_1, n_2, ..., n_d)$ and $n' = (n_1, n_2 + 1, ..., n_d)$, respectively. This is called a *basic search graph.* The neighboring relation of this graph is similar to the one that the configuration graph has.

3 The A^* Algorithm

In brief, A^* is an ordered best-first search algorithm that always examines the successors of the "most promising" node based on the function: $f(n) = g(n) + h(n)$, where $g(n)$ is the length of the shortest path from the starting node to n. The *heuristic*, $h(n)$, is an estimate of the length of the shortest path from n to any goal node.

The A^* algorithm maintains two sets of nodes: a tree of paths already obtained, called CLOSED, and a priority queue containing a subset of the leaves of CLOSED, called OPEN. After removing a node n from OPEN, it is tested whether the robot, located at n, contacts or intersects with the obstacles, i.e, the collision test is done. If n belongs to C_{free}, then it is placed on CLOSED.

3.1 On A^* Theory

Using A^* enables us to get shorter path planning times by constructing "better" heuristics for the search. If we have a better heuristic, then we know that the A^* using it is also "better".

Definition 1. [16] p. 77. A heuristic function h is *admissible* if it underestimates the cost of the optimal solution from a node n to the goal, $h(n)^*$, i.e. $h(n) \leq h(n)^* \quad \forall n$.

Definition 2. [16] p. 83. A heuristic function $h(n)$ is *monotone* if it satisfies the triangle inequality: $h(n) \leq c(n, n') + h(n') \quad \forall (n, n') \mid n' \in succ(n)$, where $c(n, n')$ is the cost of the edge from n to n'.

Theorem 1. [16] pp. 82-83. Monotonicity implies admissibility.

Definition 3. [16] p. 85. An algorithm A_2^* *largely dominates* A_1^* if every node expanded by A_2^* is also expanded by A_1^* except, perhaps, some nodes for which $f(n) = C^*$, where C^* is the cost of the optimal solution.

Theorem 2. [16] p. 85. If $h_2 \geq h_1$ and both are monotone, then A_2^* (using h_2) largely dominates A_1^* (using h_1).

3.2 Relaxed models and admissible heuristics

This section describes one method of generating admissible heuristics, namely utilizing *relaxed models.*

Relaxed models are a well-known source of admissible heuristics [16], [18], [15], [10] and [17]. They are abstract problem descriptions generated by ignoring constraints that are present in base-level problems [10]. The intuitive reason that abstractions generate admissible heuristics is because they add short-cut solution paths by simplifying the original problem [17]. Several authors emphasize, however, that relaxed problems should be easily solvable compared with their original counterparts in order to speed up the overall computation time, [18], [16], [15], [10].

Let a search problem be a three-tuple $\langle S, c, G \rangle$, where S is a set of states; $c : S \times S \mapsto \Re$ is a positive cost function; and $G \subseteq S$ is a set of goal states. A problem *instance* is a problem together with an initial state [17]. Formal representation of a relaxed model is given in the next definition.

Definition 4. [17]. An *abstracting transformation* $\phi : S \mapsto S'$ removes certain details (e.g. constraints) from the original problem $\langle S, c, G \rangle$ and produces a *relaxed problem* or a *relaxed model* $\langle S', c', G' \rangle$ iff ϕ reduces all costs and expands all the goals:

$$(\forall s, t \in S) \; c'(\phi(s), \phi(t)) \leq c(s, t) \tag{1}$$

$$(\forall g \in G) \; \phi(g) \in G', \tag{2}$$

where $c(s, t)$ and $c'(\phi(s), \phi(t))$ are the costs of the shortest paths between the corresponding nodes. Note that an abstracting transformation does not have to be a *function*, in contrast to [17]. We will utilize this in the Section HEURISTICS below.

The next theorem shows that heuristics generated by optimizations over relaxed models are monotone and thus admissible.

Theorem 3., [18], [16]. Assign every node $n \in S$ of the original problem a heuristic $h(n)$ that is the cost of the optimal solution of a relaxed problem instance with starting node $\phi(n) \in S'$. Then $h(n)$ is monotone $\forall n \in S$.

Proof [16] p. 116: Suppose $h(n)$ and $h(n')$ are the heuristics assigned to nodes $n, n' \in S$, respectively. These heuristics are minimum costs from the corresponding nodes $\phi(n), \phi(n') \in S'$ to a goal $\phi(g) \in G'$. Thus $h(n) \leq c'(\phi(n), \phi(n')) + h(n')$, where $c'(\phi(n), \phi(n'))$ is the relaxed cost, since otherwise $c'(\phi(n), \phi(n')) + h(n')$ would constitute the optimal cost from $\phi(n)$. Monotonicity follows from equation (4): $c(n, n') \leq c'(\phi(n), \phi(n'))$ $\Box$

To satisfy the conditions of Definition 4, it suffices that s and t are **neighboring nodes** in the original problem S. Finally, it is easy to prove the following theorem:

Theorem 4. If $\langle S'', c'', G'' \rangle$ is a relaxed model of $\langle S', c', G' \rangle$ and $\langle S', c', G' \rangle$ is a relaxed model of $\langle S, c, G \rangle$, then $\langle S'', c'', G'' \rangle$ is a relaxed model of $\langle S, c, G \rangle$.

4 The Multiresolution Idea

4.1 "Basic" A^* -Version

In the following, $n \in Z^d$ refers to the node whose successors, $succ(n) \in Z^d$, A^* is expanding. In short, let $n' = succ(n)$. In the "basic" A^* -application, all n's form a subset of the immediate neighbors of n (there is an edge connecting n and n' in a search graph). The costs between the n and n' are: $c(n, n') = 1 \; \forall n, n'$. Figure 2 A illustrates a 2-dimensional search graph without any obstacles. We will call it a *basic search graph.*

Figure 3 A shows a 2 DOF path planning example produced by the above basic A^* algorithm. In Fig. 3 A, gray areas are C -obstacles. Thin black lines illustrate the tree of the path candidates expanded by A^*. The path from start to goal is the bold black line. The goal is at the right end of the path. Small black dots are the expanded nodes (that are in CLOSED). In Fig. 3 A, there are 2429 nodes in CLOSED and the same amount of collision tests completed. The collision test is the most time consuming single operation during the search.

We have found this method to produce long searching times for realistic path planning tasks, hours or so, see e.g. [6].

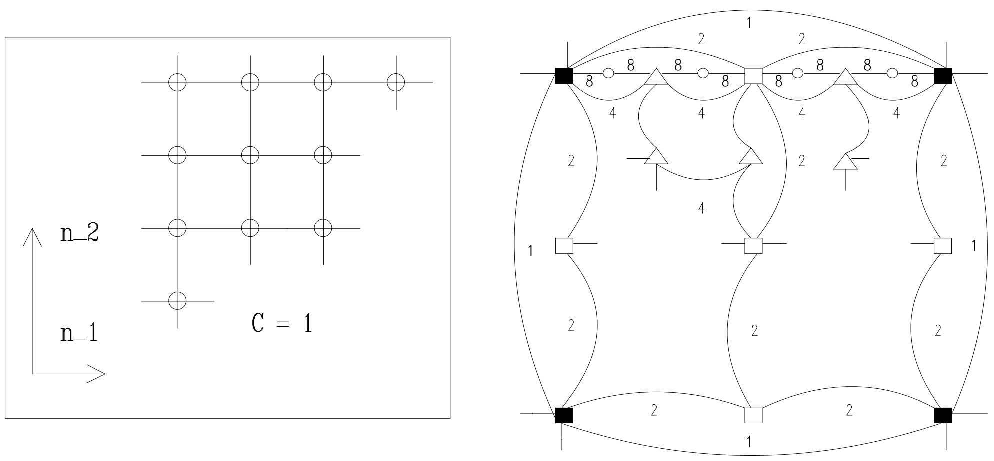

Figure 2: Fig. 2 A (left): Basic search graph, Fig. 2 B (right): Modified search graph

4.2 Modified Search Graph

Figure 2 B shows another 2-dimensional graph. We will call it a *modified search graph*. It is similar to the one in Fig. 2 A except it has many additional edges between the nodes. In Fig. 2 B, all the edges are present only in the upmost line of the nodes, for clarity.

In Fig. 2 B, the nodes $n = (n_1, n_2) \in Z^2$ marked with black squares have all their components, n_1 and n_2, divisible by 8. We say that their *m-value* is 8. The m-value is synonymous with the *resolution* of the node. The nodes with white squares have all their components divisible by 4 and *not* by 8. Their resolution is thus 4, i.e. $m = 4$. Further, all the components of the nodes with white triangles are divisible by 2 and not by 8 and 4 ($m = 2$). In general, we define the resolution or the m-value of the node n as the greatest common divisor of $n_1, .., n_d$ ($n = (n_1, .., n_d)$).

The costs $c(n, n')$, are *not* measured by $dist(n, n') =\mid n - n' \mid$. We calculated them as follows. Let the maximum allowed m -value or resolution be max. In the multiresolution algorithm, m will always be 2^k ($k = 0, 1, ..., max_k$). For example, in Fig. 2 B, $max = 8$. If $dist(n, n') = max = 8$, then $c(n, n') = 2^0 = 1$, see the black square -nodes. If $dist(n, n') = 4$, then $c(n, n') = 2^1 = 2$, see the white square -nodes. If $dist(n, n') = 2$, then $c(n, n') = 2^2 = 4$, see the white triangle -nodes, etc. The formula for the costs is: $c(n, n') = max/dist(n, n')$.

Actually, the modified search graph is a relaxed model of the graph that the multiresolution A^* expands. This is because all the edges between the neighboring nodes are not present, or the corresponding costs are infinite, when the multiresolution algorithm is running. We will discuss this next.

4.3 Multiresolution A^*

In the multiresolution search, the decision whether a configuration $q_j \in C_{free}$ *will or will not* be mapped onto the search node $n_j \in Z^d$ depends on how close that configuration is to the C-obstacles. See Figure 3 B and compare it to Fig. 3 A.

When a configuration q corresponding to the node n is far away from C -obstacles, then its successors n' are also far away from n. These distances are measured by the number of free configurations between n and C-obstacle surfaces, or n and n'. This means that only low cost edges of the graph shown in Fig. 2 B are present.

Instead, if the node n is near C -obstacle surfaces, then its successors are also near n. Now, high cost edges shown in Fig. 2 B are also present. So, the modified search graph, discussed earlier, is a relaxed model of the graph the multiresolution search expands. This is because some edges present in the modified search graph are now missing.

Figure 3 B has been produced by the algorithm based on these principles. In Fig. 3 B, there are 339 nodes in CLOSED and 1737 collision tests completed cf. Fig. 3 A (2429).

Why are we doing the multiresolution search ? On average, there are some large obstacles in C_{obst} and large connected areas in C_{free}, see e.g. Figures 1 and 3. We want to "delay" the expansion of the nodes near the C -obstacle surfaces. This is done by assigning the corresponding edges higher costs. It follows that if a path exists that goes further away from the C -obstacle surfaces, then it may be found sooner. The second reason is that the collision testing algorithms are usually slower when a node (the configuration) is near the C -obstacle surfaces.

The multiresolution A^* always finds a path from the start to the goal if one exists on a basic search graph. We will prove this in SUCCESSOR GENERATING RULES below.

However, the problem that the basic A^* solves is different from the one the multiresolution A^* solves. Thus, these algorithms can only be compared with each other empirically.

4.4 Successor Generating Rules

There are two rules to implement the multiresolution search. Their purpose is to produce the successors for the current node n, i.e. to expand n. Then A^* inserts n into CLOSED and the successors into OPEN. All the simulation results have been produced by A^* based on these rules.

There is one more notion associated with a node, in addition to its m-value or resolution. Remember that m is the greatest common divisor of $n_1, .., n_d$ where the node $n = (n_1, .., n_d)$. We define a *j-value* of the node

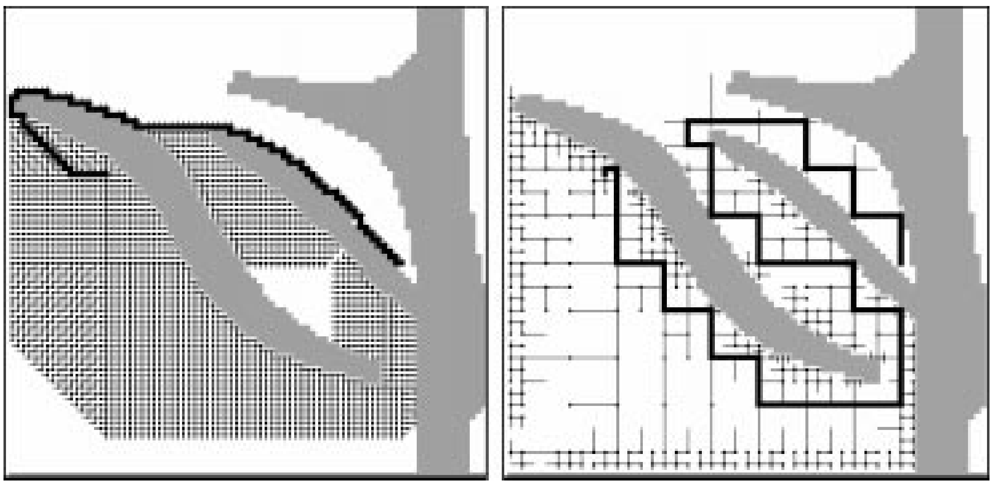

Figure 3: Fig. 3 A (left): Basic A^*, Fig. 3 B (right): Multiresolution A^*

as follows: j is the number of the components $n_1, .., n_d$ that are divisible by $m+1$. Intuitively the j-value measures how close the node is to a node whose resolution is $m+1$.

The next pseudo code generates a subset of the successors n' of n. Note the following comments. $n = (n_1, n_2, ..., n_d)$ as before. Let $k = 1, ..., d$ represent the normalized direction vector $(0, .., 1, .., 0)$, where the index of the number "1" is k. $k+1$ represents the vector, where the index of "1" is $k+1$, etc. The subroutine "PROBE(k, t, n)" returns t that is the number of the free configurations ($q \in C_{free}$) between n and a C-obstacle along the direction k. $t = 0, ..., n.m$, where $n.m$ is the resolution of n. In the next example, the maximum allowed resolution is 8. The j-value of the node n is $n.j$.

The goal is at the origo, so its resolution is $max = 8$ The first subroutine call is: "RULE-1(1, n, n'-set)".

RULE-1(k, n, n'-set):

```
(1) PROBE(k, t, n)
(2) IF (t = 0)  THEN
        n' = (n_1,..,n_i+1,..,n_d) for all i
        orthogonal to k such that n' is in C_free;
(3) ELSE IF (t = 1)  THEN n' = (n_1,..,n_k+1,..,n_d);
(4) ELSE IF (2 <= t < 4) THEN n' = (n_1,..,n_k+2,..,n_d);
(5) ELSE IF (4 <= t < 8) THEN n' = (n_1,..,n_k+4,..,n_d);
(6) END IF
(7) Determine n'.m;
(8) RULE-1(k+1, n, n'-set);
(9) Put all n' into the n'-set
```

```
        if they are not already there;
```

Line (2) esures that all the surface nodes of any big C-obstacle will be expanded eventially. While k increases from 1 to d, there is a variable t_{min} that records the minimum of the t-values. Now, let the maximum allowed m-value or resolution be max. While **RULE-1** decreases the resolutions of the successor nodes, **RULE-2** tries to increase them.

RULE-2(t_{min}, **k, n,** $n' - set$):

```
(10) IF (n.m = max) THEN
(11)    n' = (n_1,..,n_i+max,..,n_d) for all i
        such that n' is in C_free,
        Set n'.m = max;
(12) ELSE
(13)   IF (t_min = n.m) THEN
(14)      n' = (n_1,..,n_i+n.m,..,n_d) for all i,
          Determine n'.j and n'.m,
          Accept n' if (n'.j > n.j);
(15)   IF (t_min < n.m) THEN
(16)      n' = (n_1,..,n_i+n.m,..,n_d) for all i <> k
          such that n' is in C_free,
          Determine n'.m;
(17) END IF
(18) Include all n' into the n'-set
     if they are not already there;
```

Finally, the costs $c(n', n)$ are assigned in the same way as was done in MODIFIED SEARCH GRAPH.

Theorem 5. The A^* using the Rules 1 and 2 finds a path to the goal if it exists.

Proof: A path must always go through nodes whose successors either collide with the C-obstacles or not. The algorithm finds subpaths of the latter type because the j-values of the nodes always increase, see line (14). This implies that also the m-values of the nodes along such subpaths increase until the maximum is reached. This maximum is also the resolution of the goal. Then the search proceeds similarly to the basic A^*, lines (10-11).

Lines (2-5) enables the paths to approach to the C-obstacle surfaces until their nodes' successors collide. Line (2) guarantee that all these surface nodes will be expanded eventially. So, the surface of any big C-obstacle will be explored by the algorithm, if necessary. Lines (13-16) guarantee that the paths can escape from the C-obstacle surfaces to the "free" space again. □

5 Heuristics

In this section, we will present admissible heuristics for the multiresolution algorithm and use them in the experiments of the next section. First, we will introduce a method of decomposing a d -dimensional problem into several lower dimensional problems that together form a relaxed model of the former one.

5.1 Constructing ϕ Using Projections

Assume a problem $\langle N, c, G\rangle$ whose nodes $n \in N$ and $g \in G$ are vectors in Z^d. We will construct a relaxed problem $\langle N', c', G'\rangle$ as follows.

We define an abstracting transformation $\phi : N \mapsto N'$: $\phi(\pi_j(n)) = \pi_j(n)$, where $n \in N$ and $\pi_j(\cdot) \in N'$ $(j = 0, 1, 2, ..., d)$. $\pi_j(\cdot)$ is a *projection*. For example, if $d = 2$ then $\pi_1((n_1, n_2)) = (n_1, s_2)$ and $\pi_2((n_1, n_2)) = (s_1, n_2)$ $(s \in Z^d)$. Note that ϕ is a function of π_j but *not* necessarily a function of n. If we accept the latter, then we can do the following.

Let us consider a 2-dimensional example. Let a node $n = (n_1, n_2) \in N$, the start node $s = (s_1, s_2) \in N$, and the goal node $g = (g_1, g_2) \in G$. We define $\pi_1((n_1, n_2)) = (n_1, s_2) \in N'$ and $\pi_2((n_1, n_2)) = (s_1, n_2) \in N'$ for all nodes in N. It follows that the start node s remains the same. In particular, $\pi_1(g) = (g_1, s_2) \in N'$ and $\pi_2(g) = (s_1, g_2) \in N'$. The goal node stays where it was, in other words, we define an additional mapping $G' \ni \pi_0(g) = g$, where the argument $g \in G$.

Next, we will define the costs c' between the nodes $\pi_j(n_1)$ and $\pi_j(n_2)$, where n_1 and n_2 are now vectors in N (Z^2). First, if the first coordinate of n_1 and n_2 is the same, say r, then $c'(\pi_1(n_1), \pi_1(n_2)) = 0$, since they have been mapped onto the same node (r, s_2). Second, if $n_1 = (s, v)$ and $n_2 = (t, w)$ so that $t \neq s$, and the nodes are neighbors in N, then $c'(\pi_1(n_1), \pi_1(n_2)) = min\{c(n_1, n_2)\}$. In this paper, also $v = w$.

Similarly, if the second coordinate of n_1 and n_2 is the same, say r, then $c'(\pi_2(n_1), \pi_2(n_2)) = 0$, since they have been mapped onto the same node (s_1, r). If $n_1 = (v, s)$ and $n_2 = (w, t)$ so that $t \neq s$, and the nodes are neighbors in N, then $c'(\pi_2(n_1), \pi_2(n_2)) = min\{c(n_1, n_2)\}$.

Finally, there remains the question of how to define the costs $c'_1 = c'(\pi_0(g), (g_1, s_2))$ and $c'_2 = c'(\pi_0(g), (s_1, g_2))$. Let us find the minimum lengths of the routes from s to (g_1, s_2) and (s_1, g_2) by searching and call them t_2 and t_1, respectively. Then we assign $c'_1 = t_1$ and $c'_2 = t_2$.

We have now constructed a transformation ϕ that satisfies Definition 4 and, as a result, we have a relaxed model $\langle N', c', G'\rangle$ of $\langle N, c, G\rangle$. The interpretation is that we have decomposed a two-dimensional problem $\langle N, c, G\rangle$ into two one-dimensional subproblems called together $\langle N', c', G'\rangle$. Solving them for a particular node $n' \in N'$ essentially needs two one-dimensional searches and is computationally effective.

The above argument that we call here the **projection principle** can easily be extended to higher dimensional search spaces.

We calculate the heuristic for $\langle N, c, G\rangle$by by summing the path lengths obtained by solving the subproblems. This is the same as solving the relaxed problem $\langle N', c', G'\rangle$. It follows from Theorem 3 that the heuristic is monotone and thus admissible.

However, we must be sure that the optimum paths for the subproblems *can be found.* This usually means that the problem $\langle N, c, G\rangle$ *already is* a relaxed model of the original problem, constructed in a proper way. We will discuss this next.

5.2 Manhattan -maxres

This heuristic is like the Manhattan distance from a node n to the goal that is most often used in conjunction with the basic A^*.

Suppose the robot manipulator has d degrees-of-freedom (DOF). The **Manhattan distance** between a node n and the goal node g is: $\|n-g\|_{L^1} = \sum_{i=1}^{d} \mid n_i - g_i \mid$, where $n, g \in Z^d$, i.e., n_is and g_is are integers. We cannot use the Manhattan distance directly because the distance between a node and its successor depends on the node's resolution (the step size). Therefore we must use the maximum resolution value, max, see MODIFIED SEARCH GRAPH.

This is done as follows. First, every $\mid n_i - g_i \mid$ -term is divided by max. Second, if there is a non-zero reminder, then 1 is added to the result. For example, if $max = 4$ and $\mid n_i - g_i \mid = 10$, then $h_i = 2+1 = 3$. The **Manhattan-maxres** -heuristic is the sum of these h_i-values: $h(n) = \sum_{i=1}^{d} h_i$. One can argue that when max is high then $h(n)$ may not be very effective because the actual path lengths may be very long compared to it.

Using the Manhattan distance as a heuristics corresponds to solving a relaxed problem of the base level problem $\langle S, c, G\rangle$: all the C-obstacles have been ignored. Let us call it $\langle S', c', G'\rangle$ that now equals $\langle N, c, G\rangle$ discussed above.

Calculating the Manhattan distance corresponds to solving d one-dimensional subproblems called together $\langle N', c', G'\rangle$. They can be constructed from $\langle N, c, G\rangle$ by using the projection principle (in d dimensions). It follows from Theorems 3 and 4 that the Manhattan distance heuristic is admissible.

5.3 Manhattan-multires

The Manhattan-maxres heuristic may be very uninformative if the maximum resolution max is high. To utilize more our multiresolution algorithm, we will construct a better heuristic. Consider a d-dimensional version of the *modified search graph* in Fig. 2 B. Remember, it is a relaxed model of the graph that the multiresolution algorithm expands during the search. This is since all the obstacles have been ignored. So, we call it $\langle S', c', G' \rangle$ that equals $\langle N, c, G \rangle$.

Next, we apply the projection principle to it and obtain d one-dimensional subproblems called together $\langle N', c', G' \rangle$. We get the **Manhattan-multires** heuristic by solving these separately and summing the optimum path lengths. Again, we conclude from Theorems 3 and 4 that the heuristic is admissible.

However, solving each one-dimensional subproblem needs one-dimensional searching, in contrast to the Manhattan-maxres heuristic. This searching is done by the multiresolution algorithm. The algorithm always finds the optimum path because there are no obstacles involved. The Manhattan-multires heuristic is always bigger or as big as the Manhattan-maxres for every node and thus is better, see ON A^* THEORY.

5.4 K-DOF -multires

Consider again the d-dimensional version of the *modified search graph* in Fig. 2 B. We decompose it into two subspaces: the first one is k-dimensional ($k \leq d$) and the second one is $d-k$-dimensional. The first subspace corresponds to the robot's k first degrees-of-freedom (DOFs) and the second one the $d-k$ rest DOFs.

First, we ignore the obstacles in the second subspace and calculate the Manhattan-multires heuristic in it, as was done previously. Let us call this subspace Z_2^{d-k}. Suppose $Z^d \ni n = (n^k, n^{d-k})$, where $n^{d-k} \in Z_2^{d-k}$. Then, let $h(n^{d-k})$ denote the Manhattan multires-heuristic for n^{d-k}.

Second, we preserve the obstacles in the first subspace but ignore the $d-k$ last degrees of freedom of the robot operating there. If we have, e.g. a usual 5- or 6-DOF industrial robot, then we ignore the 2 or 3 last DOFs, respectively. As a result, we have a robot manipulator that is "amputated" above the wrist.

Third, we simplify the "amputated" robot by modeling it only by a couple of polygons that are inside the volume covered by the original robot. The original robot model usually consists of tens of polygons. This makes the collision tests more effective. Let us call this simplified robot model a *reduced robot.*

Fourth, we do collision tests by using the reduced robot. Some configurations that originally are in C_{obst}, by using the original robot, may now be in C_{free}. This is because the volume covered by the reduced robot is a subset of the volume covered by the original robot. It means that some costs between the neighboring nodes decrease from infinity to a small number. Also, since we only have a k-DOF robot, the optimum path can be found by searching in the first subspace. Let us call this subspace Z_1^k. Suppose $Z^d \ni n = (n^k, n^{d-k})$, where $n^k \in Z_1^k$. Then, let $h(n^k)$ denote the heuristic for n^k.

Obviously, the space $Z_1^k \times Z_2^{d-k}$ is a relaxed model of the modified search graph. The heuristic, called the **k-DOF-multires** is: $h(n) = h(n^k) + h(n^{d-k})$. We can now apply the projection principle to show that this heuristic is admissible.

Let $Z_1^k \times Z_2^{d-k}$ correspond to $\langle N, c, G\rangle$. To construct a relaxed model $\langle N', c', G'\rangle$, we define the projections $\pi_j(\cdot)$ $j = 0, 1, ..., d-k+1$ as follows. For example, let $k = 3$ and $d = 5$. Let a node $n = (n_1, n_2, ..., n_5) \in N$, the start node $s = (s_1, s_2, ..., s_5) \in N$ and the goal node $g = (g_1, g_2, ..., g_5) \in G$. First, $G' \ni \pi_0(g) = g$, as before. Second, $\pi_1(n) = (n_1, n_2, n_3, s_4, s_5) \in N'$. Third, $\pi_2(n) = (s_1, s_2, s_3, n_4, s_5) \in N'$ and $\pi_3(n) = (s_1, s_2, s_3, s_4, n_5) \in N'$. $\pi_1(n)$ projects every n onto the 3-dimensional subspace Z_1^3. $\pi_2(n)$ and $\pi_3(n)$ project every n onto one-dimensional subspaces that together form Z_2^{5-3}. The costs are defined analogously to the projection principle example, see HEURISTICS.

So, we have constructed $\langle N', c', G'\rangle$, a relaxed model of $\langle N, c, G\rangle$ that further is a relaxed model of the modified search graph. It follows from Theorems 3 and 4 that the k-DOF-multires -heuristic is admissible. The k-DOF-multires heuristic is always bigger or as big as the Manhattan-multires for every node and thus is better than it.

Actually, we calculate the $h(n^3)$ -part of the heuristic for every n by the breadth-first search in Z_1^3 starting from the "projected goal" $(g_1, g_2, g_3, -, -)$. The search is done by our multiresolution algorithm in 3 dimensions. The calculations were done off-line i.e. before starting the multiresolution search of the original problem. The overhead time for doing this is about 1.5 minutes ($k = 3$).

Utilizing the k first DOFs of the robot to guide the search process is not new in robotics, see e.g. [14]. However, we have applied it in conjunction with our multiresolution A^* algorithm and shown that the resulting heuristic is admissible.

6 Experimental Results

6.1 Test Cell

Figure 4 shows the robot test cell. It has been adapted from [11]. The actual dimensions may differ from those in the reference. There is a 5-DOF Adept industrial robot with an L-shaped object attached to its gripper. The left picture shows the starting configuration and the right picture the goal.

The robot has to bend its wrist to get the L-shaped object out of the wicket. Then it has to lift its linear axis and to rotate its first and second

	TIME	COLL.	CLOSED		TIME	COLL.	CLOSED
2.1	1609	354960	50605	2.1	749	166572	27267
2.2	592	159350	8428	2.2	152	39418	2325
2.3	387	107327	5057	2.3	101	25689	1614

Table 1: Table 1 A (left): max = 8, Table 1 B (right): max = 16

link to avoid the middle block, see Figure 5.

6.2 Simulations

We record the program execution times, the number of the collision tests and the number of the nodes in CLOSED. The times reported are CPU times in seconds on a SGI Indigo 2 computer with 150 Mhz R4400 processor and 96 megabytes of main memory. The collision test was done using a software package called *RAPID* (Rapid and Accurate Polygon Interference Detection) [9]. The collision test is the most expensive single operation in the search process. The discretizations for the Adept robot are 150x100x40x100x40 corresponding to about 2 *cm* maximum movement of the robot (in W).

We have used four algorithm versions in the tests. The first one is the basic A^*. It has a heuristic that is similar to the 3-DOF-multires heuristic. The basic A^* was very slow: we had to stop the searching since the main memory was full. This happened after about 1.5 hours, 970000 collision tests, 590000 nodes in CLOSED.

The second version is the new multiresolution A^* algorithm. It has three variants corresponding to the three heuristics presented. The first one uses the Manhattan-maxres heuristic (**algorithm 2.1**). The next one uses the Manhattan-multires heuristic (**algorithm 2.2**). The third one uses the 3-DOF-multires heuristic (**algorithm 2.3**). The off-line calculation time for the 3-DOF-multires heuristics was about 1.5 minutes. This is not included in the results since these calculations can be utilized for many different path planning problems in the same robot cell.

Tables 1-2 show the simulation results. The first column of each table, e.g. Table 1 A, shows the algorithm version, the second one the planning times in seconds, the third one the number of the collision tests, the fourth one the nodes in CLOSED.

7 Conclusions

Tables 1-4 show that the new multiresolution idea, embedded in the structure of A^* algorithm, clearly outperformed the basic A^* search that had to

	TIME	COLL.	CLOSED		TIME	COLL.	CLOSED
2.1	1921	428770	48957	2.1	>2464	>500000	> 50000
2.2	353	95333	3466	2.2	487	118875	3929
2.3	333	92379	3264	2.3	441	115891	3754

Table 2: Table 2 A (left): max = 32, Table 2 B (right): max = 64

be stopped during the search.

The computational costs are comparable to the measured computing times, and the number of the collision tests. The amount of the memory used, on the other hand, is comparable to the amount of the nodes in CLOSED.

In this simulation example, the best searching times did not vary very much as a function of the maximum m -value, max, the maximum allowed resolution, see the last lines of Tables 1 A, 2 A, and 2 B. An exception is Table 1 B ($max = 16$), which shows the shortest planning times of all.

The Manhattan-maxres -heuristic produced searching times that were longer than with the two other heuristics: the Manhattan-multires and the k-DOF-multires. The amounts of the memory used with the Manhattan-maxres were also clearly biggest. We stopped the test run shown in Table 2 B, the second row, after 2464 seconds.

On the other hand, the 3-DOF -multires -heuristic was not much better than the Manhattan-multires. Tables 2 A and 2 B show insignificant differences, the reason of which is unclear at the moment.

On the whole, the searching times of the multiresolution algorithm seem to be reasonably short for on-line path planning in low dimensional robot configuration spaces, $d = 5$.

7.1 Future Work

We are now studying whether the heuristics obtained by sampling the search space with a coarse resolution can be shown to be admissible. We are also developing more sophisticated rules for successor generation to further decrease the number of the successors of the nodes in CLOSED.

Acknowledgements:
This work was supported by the Academy of Finland.

References

[1] Michael Barbehenn and Seth Hutchinson. Efficient search and hierarchical motion planning using dynamic single-source shortest paths trees. In *IEEE International Conference on Robotics and Automation*, May 1993.

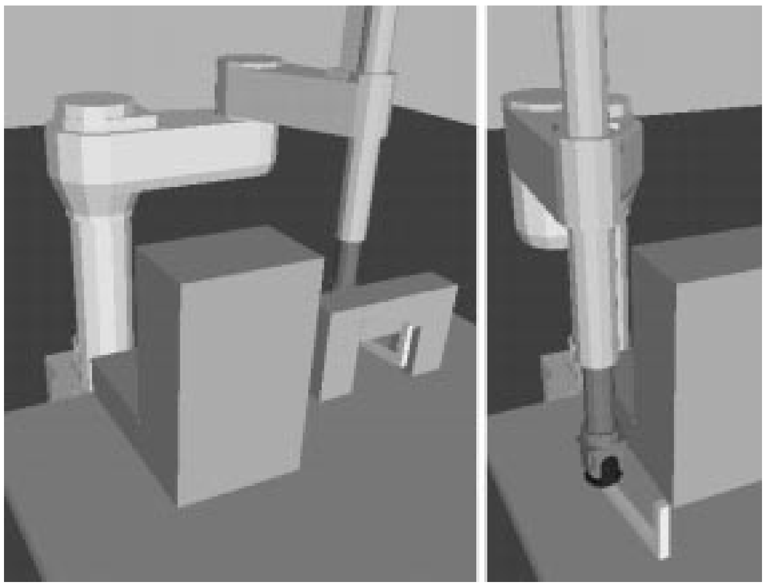

Figure 4: Robot test cell

[2] Michael Barbehenn and Seth Hutchinson. Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees. *IEEE Trans. Rob. Autom.*, 11(2):198–214, 1995.

[3] Jerome Barraquad and Jean-Claude Latombe. Robot motion planning: Distributed representation approach. *The International Journal of Robotic Research*, 10(6):628–649, 1991.

[4] Bernhard Clavina. Solving findpath by combination of goal-directed and randomized search. In *IEEE International Conference on Robotics and Automation*, 1990.

[5] Christopher I. Conolly. Harmonic functions and collision probabilities. In *IEEE International Conference on Robotics and Automation*, Vol.4 May 1994.

[6] Antti Autere et. al. Robot motion planning by enhanced a^* algorithm. In *27th International Symposium on Industrial Robots (ISIR)*, October 6-8 1996.

[7] Tomas Lozano-Peres et. al. Spatial planning: A configuration space approach. *IEEE Trans. Computers C-32(2):108-120*, 1983.

[8] Tomas Lozano-Peres et. al. A simple motion planning algorithm for general robot manipulators. In *5th AAAI, Philadelphia*, 1986.

[9] S. Gottschalk et.al. Obbtree: A hierarchical structure for rapid interference detection. In *Computer Graphics Proceedings, SIGGRAPH 96*, August 4-9 1996.

[10] Othar Hansson, Andrew Mayer, and Marco Valtorta. A new result on the complexity of heuristic estimates for the a^* algorithm. *Artificial Intelligence*, 55:129–143, 1992.

[11] Y. K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computing Surveys*, 24(3):219–284, 1992.

[12] Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for path planning: articulated robots. In *IEEE/RSJ/GI International Conference on Intelligent Robotics and Systems (IROS'94)*, Vol.3 September 1994.

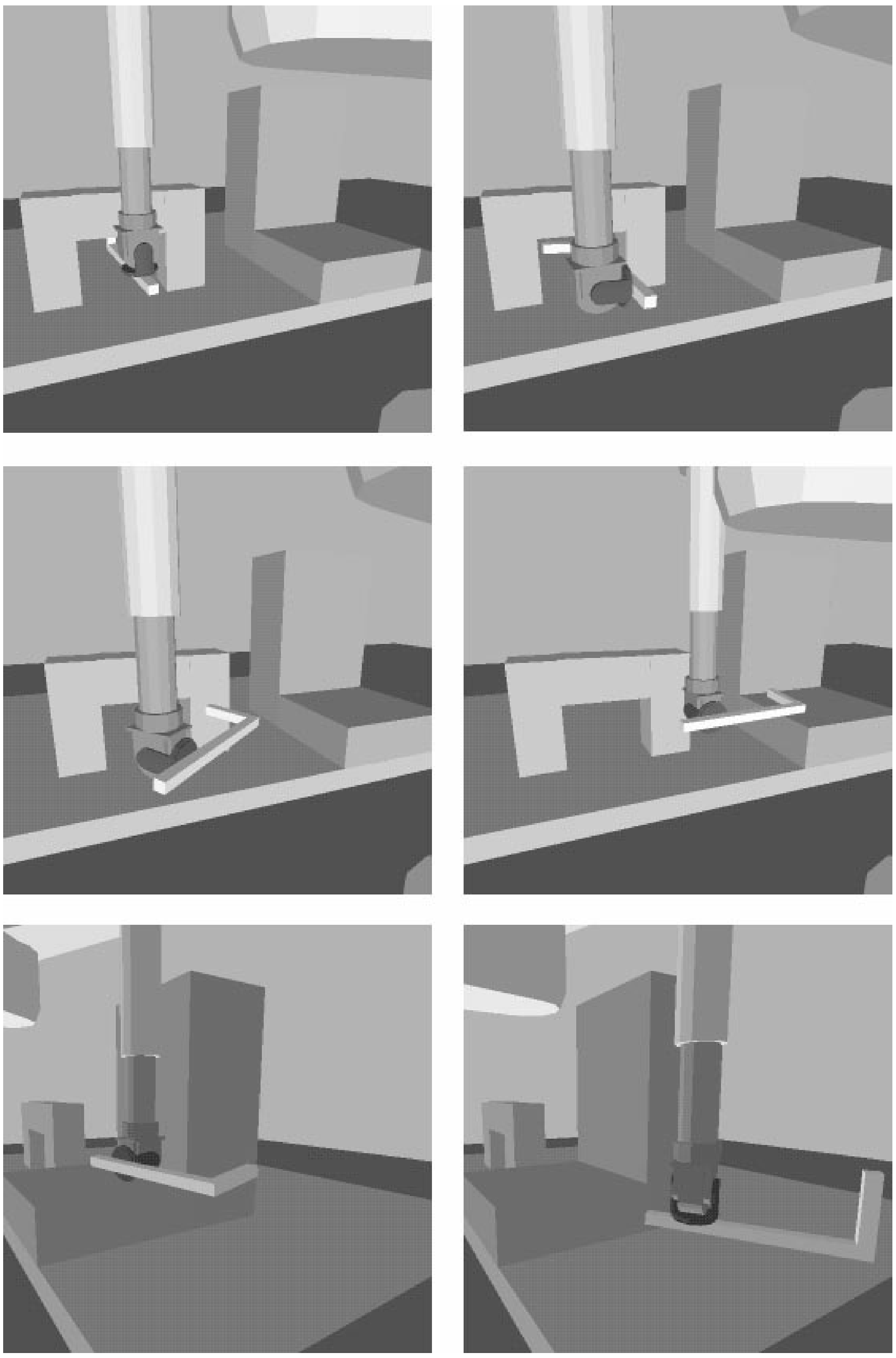

Figure 5: The resulting path

[13] Lydia Kavraki and Jean-Claude Latombe. Randomized preprosessing of configuration space for fast path planning. In *IEEE International Conference on Robotics and Automation*, Vol.3 May 1994.

[14] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[15] Jack Mostow and Armand E. Prieditis. Discovering admissible heuristics by abstracting and optimizing: A transformational approach. In *IJCAI-89*, pages 701–707, 1989.

[16] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, 1984.

[17] Armand Prieditis and Robert Davis. Quantitatively relating abstractness to the accuracy of admissible heuristics. *Artificial Intelligence*, 74:165–175, 1995.

[18] Marco Valtorta. A result on the computational complexity of heuristic estimates for the a^* algorithm. *Information Sciences*, 34:47–59, 1984.

[19] Charles W. Warren. Fast path planning using modified a^* method. In *IEEE International Conference on Robotics and Automation*, May 1993.