Mobility Analysis for Feasibility Studies in CAD Models of Industrial Environments

Carl van Geem, Thierry Simeon, Jean-Paul Laumond, J.-L. Bouchet, J.-F. Rit

Mobility Analysis for Feasibility Studies in CAD Models of Industrial Environments

C. Van Geem*, T. Siméon, J-P. Laumond
LAAS-CNRS, Toulouse, France
J-L. Bouchet, J-F. Rit
EDF-DER, Chatou, France

Abstract

This paper presents a variant of a probabilist approach to motion planning. The objective is to face large, complex industrial environments for maintenance purpose. The contribution is based on a structuring of the workspace into boxes that limits the size of the graph to be searched. The algorithm was integrated in a commercial CAD system and real-size experiments are presented.

1 Introduction

Feasibility studies for the planning of maintainance interventions at industrial sites are often done using CAD systems. Scenes have high complexity (more than 10000 polyhedral facets). Commonly devices like carts and cranes (with up to 4 DOFs, occasionally up to 8 DOFs, a tree-like kinematic structure) intervene and must carry objects through the scene. Searching for collision free motions through densely populated scenes tried by hand is a difficult and time consuming task. Automated mobility analysis therefore would considerably help to improve quality and to reduce the cost of such feasibility studies.

We chose to implement a probabilistic path planner constructing a connectivity graph in configuration space (C-Space), for the following reasons. Firstly, that approach plans a path for a point in C-Space and thus is independant as a method from the particularities of the device. Specificities of the device can be expressed by the use of a particular strategy (local planner) for the connection of two configurations in the graph. Due to the modularity of the probabilistic path planning approach implementing and adding another local planner is an easy programming task.

Due to the high complexity of the industrial scenes we concentrated our efforts on the strict limitation of the size of the graph and the costly operations like collision detection and search for connections between nodes in the graph. We reject obsolete nodes corresponding to easy to attain parts, and aim at attaining narrow passages in the Workspace. A graph consisting of a forest of trees suffices to capture connectivity of C-Space.

In recent years the Probabilistic Roadmap approach has become a popular method for path planning problems ([7], [2]), and first results of the analysis of the method's behavior exist ([6]). The performance of the method can be increased by strategies which guide the algorithm in its random search for the connectivity properties of free C-Space ([5], [1]). A first step toward the application of the method in evolving industrial environments was done in [4] and [3], as well as in [8] where an off-line pre-computed roadmap is updated during execution of a path.

The contributions presented here are the use of a partitioning of Workspace in boxes (Section 3) in order to filter and select nodes on which costly operations are executed, the use of several local methods for expressing particularities of the device and the integration of the planner in the CAD system ROBCAD, product of Tecnomatix and used by EDF (Section 4). Experimental results are presented in Section 5.

2 Algorithm

The probabilistic path planner approach consists of two phases: a learning phase and a query phase. In the learning phase a graph is constructed in C-Space. Configurations are choosen randomly. If the device in a new configuration is not in collision, the configuration is added as a node to the graph. Then the new node is connected by arcs to some other nodes in the graph if a collision free path between them is found. Such a path is searched for by a local method. Usually the choice of configurations is not entirely ran-

*supported by Electricité de France (EDF-DER)

dom, but influenced by guidelines chosen by the implementor. Moreover, usually the generated graph is refined further by some dedicated methods in order to increase its quality. In the query phase an initial and a goal configuration is connected to the graph using a local planner, and the solution is obtained by a graph search. The graph can be re-used for several queries.

The main steps of our implementation are illustrated in the pseudo-code below.

```
1. PartitionWorkspace();
2. repeat {
        ExtendGraphRandomly();
        RefinePoorlyVisitedBoxes();
        RefineUnreachedBoxes();
        RefineSmallTrees();
}
until( GraphConstructionCriterion() )
3. Query();
4. EnhanceQuality();
```

In the learning phase the choice of configurations and the refinement is based upon a precomputed partitioning of the Workspace in boxes (see Section 3 for details). In the learning phase calls to random generation of nodes and calls for refinement of the graph constructed so far alternate until the graph reaches a certain quality. The loop in step 2 is stopped based upon an analysis of the evolution of the number of trees in the graph as well as their relative sizes. When a query resulted in a path, the quality of the path initially found can be enhanced using an optimization routine. Each of the four steps are accessible seperately by the end-user through the interface.

3 Boxes and Learning Phase

Preprocessing the Workspace: As a preprocessing step, we partition Workspace in boxes of different sizes (PartitionWorkspace()). The goal is to reduce the number of poses of the device in easily reachable parts of the Workspace and to augment the number of poses in parts where obstacles are densely present and holes (like door openings) in obstacles occur. We attempt to create automatically large boxes in easily reachable areas and small boxes elsewhere.

We start off the construction of the partitioning with a bounding box around Workspace. This bounding box might be defined by the end-user using the interface, or is a huge box by default. We recursively cut the box in two smaller boxes (usually of a different size) by a cutting plane, if there are many (polyhedral) obstacles which have at least one vertex in the box. We cut alternately along the X, the Y, or the Z-direction of the global coordinate system of the scene.

In order to cut a box B in two, we must select its cutting plane. Suppose the box should be cut by a plane perpendicular to the X-axis of the global coordinate system. Then for each vertex v on the outer facets of a polyhedral obstacle, v lying in B, we consider the plane P_x^v perpendicular to the X-axis. If all vertices on the outer facets adjacent to v lie on the same side of P_x^v, we call P_x^v a candidate plane for B. We order the candidate planes P_x^v according to increasing X-coordinate value of the corresponding vertices v. The cutting plane is the middle one in this ordered list. Figure 1 illustrates this procedure.

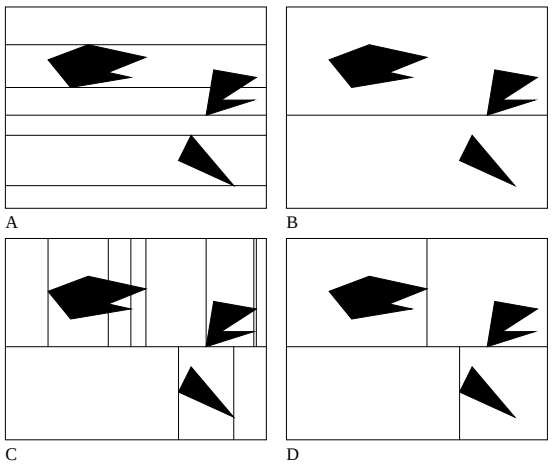

Figure 1: Candidate Planes and Cutting Planes.

The cutting is stopped once a threshold on the total number of boxes is reached, or when no box contains more than a minimal number of obstacles.

In the following we will say that 'configuration c corresponds to a box' if a user defined point on the device, usually at the end-effector, ends up being positioned in the box when the device is put in configuration c. Since a node in the graph is a configuration added to the graph, we say that 'box B contains node n' if the configuration corresponds to B. We will say that two boxes are 'neighbors' if their intersection is non-empty.

Use of Boxes in Learning Phase: We use boxes on four places in the learning phase:

Firstly, when a random configuration c is chosen (ExtendGraphRandomly()) it is first determined to which box it corresponds. If that box already contains more than M nodes, we reject c. The parameter M is increased at the end of each visit of the loop.

After a while we only test new configurations when they correspond to poorly visited boxes.

Secondly, during the first refinement step (RefinePoorlyVisitedBoxes()) we consider all nodes in boxes which contain less than m nodes). The parameter m is increased at each visit of the loop. We randomly perturbate some or all of the C-Space coordinates of these nodes. Doing so we make more nodes in probably difficult to reach areas.

Thirdly, in the refinement step we also try to attain boxes which contain no nodes (RefineUnreachedBoxes()). These boxes might not be reachable at all, or only with extreme difficulty. In the refinement step we consider all nodes in all boxes neighboring empty box B_e and from thereon we try to move toward B_e until the device is in B_e or until we give up.

Fourthly, we never call the local planner to try to connect two nodes in two different non-neighboring boxes.

Both the random generation of configurations and the refinement step tend to produce small trees. In the refinement step RefineSmallTrees(), by randomly perturbating some nodes in small trees, some small trees get merged.

Performance: In Figures 2 and 3 each dot indicates the position of a cart corresponding to a node (a configuration in C-Space). Figure 2 shows the node distribution when just ExtendGraphRandomly() (that is, without boxes) is used. Figure 3 shows the boxes and nodes generated using refinement and boxes.

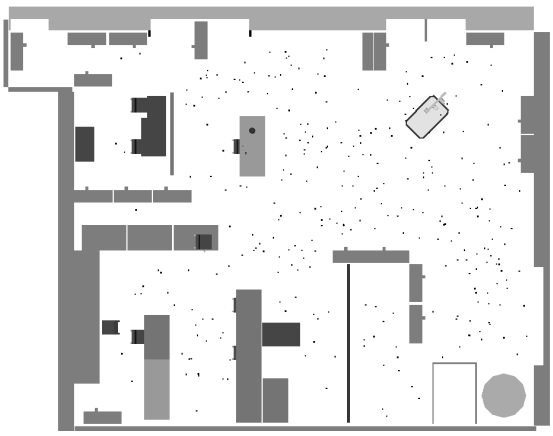

Figure 2: Node distribution (without boxes).

The table below shows for the same example that connectivity is captured better with fewer nodes and after fewer calls to local planner and collision checker.

Figure 3: Node distribution (with boxes).

	boxes	no boxes
nodes	264	300
arcs	510	578
trees	9	11
local planner	3800	5398
collision test	57454	202442

In this table: the number of nodes, arcs, trees in the graph; the number of calls to local planner and collision checker. In the case where boxes were not used, the largest tree contained 273 nodes, whereas the few other nodes were scattered about a tree of 15 nodes, a tree of 4 nodes and 8 trees of one isolated node. When boxes and refinement were used, the number of nodes per tree where 121, 45, 36, 27, 11, 11, 7, 3, and 3.

ExtendRandomly()	boxes	no boxes
generated	1998	723
filtered	534	-
accepted	84	300
Refinements		
generated	2521	-
filtered	2385	-
accepted	180	-

In this table: the number of configurations generated, filtered (by boxes, and thus tested for collision), accepted during the phase of random extension and during the refinement phases.

The table below shows the evolution of the connectivity during the graph construction for the first experiment in Section 5. In the fifth column (rej.) the number of randomly generated poses which were rejected through the filtering mechanism with boxes is listed. In the last column (lp.) the number of calls to the local planner is mentioned.

phase	nodes	arcs	trees	rej.	lp.
E	24	30	9	108	87
P	30	44	8	178	129
U	31	46	8	213	135
T	64	120	3	213	468
E	70	132	1	234	513

In this table: E for ExtendGraphRandomly(), P for RefinePoorlyVisitedBoxes(), U for RefineUnreachedBoxes(), T for RefineSmallTrees(). The number of trees decreases until connectivity of C-Space is captured (in this experiment there is just one connected component).

4 Integration into a CAD system

ROBCAD: The CAD system ROBCAD allows to design complex scenes as well as to define the kinematic properties of devices. It also provides a C-library of functions which give access to geometrical and kinematical data of the scene as well as to some of its operations. In our implementation we used the function of ROBCAD for static collision checking. Very likely the path planner would benefit from a well-performing function for distance computation, but we estimated the effort needed to implement one too big and decided for functionalities with a higher priority.

Genericity: Particularities of certain devices (for instance, for safety and controllability reasons it might not be allowed to move more than one DOF of a crane at a time), and kinematic constraints (carts, for instance) can be expressed by the use of a particular local planner. We provide a library currently containing four local planners. Through the interface, in each stage of the algorithm the user can modify the choice of the local planner used.

The first local planner (LIN) tries to connect two configurations by a simple move along a straight line in C-Space. This local planner is fast, but does not always perform well in graph construction.

The second local planner (STEP) moves only one joint at a time (a 'Manhattan walk'). Sometimes such a path can be found when a linear move in C-Space is unsuccessful. Also, using the Manhattan walk in the learning phase or for replacement of the path in the path quality enhancement step, a path is found taking into account this particularity of cranes.

A third local planner (RS) is dedicated to carts with a non-holonomic constraint. It looks for a path commonly known as a Reeds-Shepp path (made up of straight line segments and circular arcs, [9]).

The fourth local planner (POT) uses a pseudo-potential field approach. It looks for a path along the linear line in C-Space. In case of a collision along that line, the planner tries to circumvene the obstacle by a small random walk away from the straight line. This local planner is useful in the query phase when initial or goal configurations are close to an obstacle.

Quality Improvement of a Path: Once a path was found, the interface (EnhanceQuality()) allows the end-user to improve a path to the needs of the application. Two functionalities allow so.

The first tries to shorten the path. It selects randomly two points on the path and tries to connect them with the local planner choosen by the user. In case of success, the part between these two points on the curve is replaced. By repeatedly calling this procedure the path gets shorter. This is particularly true since the initial path was found in a tree.

The second tries to replace the whole path by a new path using the local planner choosen by the user. For a cart (X, Y, θ) with non-holonomic constraint any geometrical path can be transformed in a path of the type Reeds-Shepp under some often fulfilled topological condition. It therefore is useful to use a simple local planner (LIN, STEP) in learning and query phase, resulting in a geometrical path, and to replace that path afterwards using RS. For a crane it can be useful to transform a path generated with LIN by a path computed with STEP so that no two joints move simultaneously.

5 Experiments

Figure 4: Devices.

Figure 4 shows three devices on which we carried out the experiments. The first is a traveling crane with 4 DOFs (TTTR), the second a rotating crane with 4 DOFs (RTTR), the third a cart (X, Y, θ). Typically devices with such kinematic structures are used for maintenance purposes in power plants of EDF. Joints of cranes are not allowed to move but one at a time. Carts usually have non-holonomic constraints.

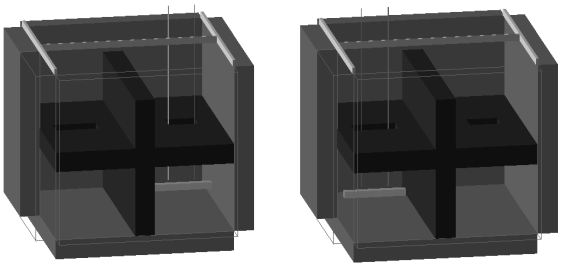

Figure 5: initial and goal pose.

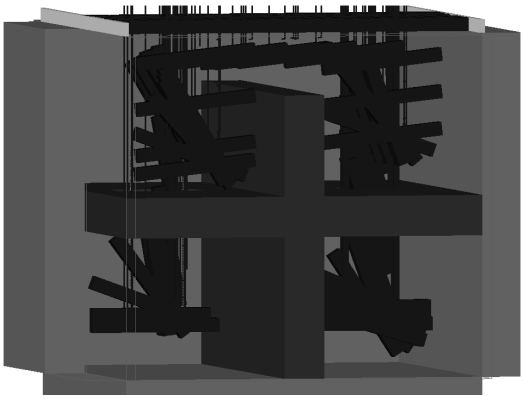

Figure 6: a path.

In the first (academical) experiment, a traveling crane must move a (heavy) load up through the hole in the middle floor on the right, and down again through the hole on the left.

Figure 5 shows initial and goal configurations. The connectivity of the scene was captured by a graph of 70 nodes (consisting of 1 tree), constructed in 170 seconds. The initial path was smoothened afterwards in 35 seconds. Figure 6 shows an example of a path found by the planner.

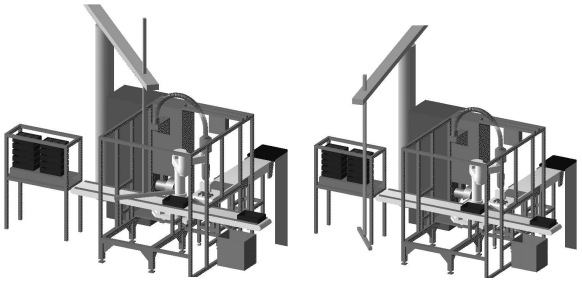

Figure 7: initial and goal pose.

In the second experiment, a rotating crane must evacuate a bar from the centre of the scene to a position on the left. Figure 7 shows initial and goal configuration. A graph was constructed until a path was found. Since our implementation uses a static collision detector along local paths, the size of a discrete step between two calls of the collision checker along a local path in this scene must be very small. The graph was constructed in 570 seconds. The path shown in Figure 8 was generated after a smoothening phase which took 40 seconds. Reusing the same graph for other queries (other initial and goal configurations) delivered paths as well except at some positions very difficult to reach.

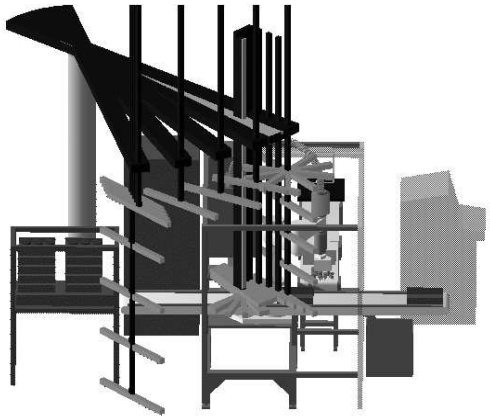

Figure 8: a path.

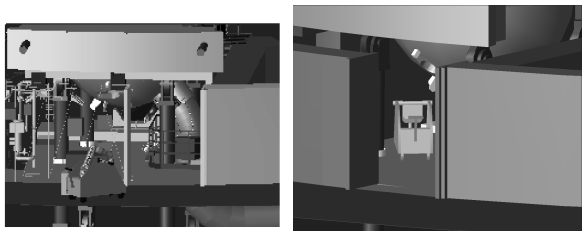

Figure 9: initial and goal pose.

In the third experiment, a cart in a nuclear power plant must enter a door and set itself underneath the steam generator. Figure 9 shows initial and goal configuration. In a simplified model of the environment (model 1) a graph was constructed until a (geometrical) path was found in 5 minutes. The transformation of the geometrical path to a path taking into account the non-holonomical constraints (using local planner RS) took less than a minute. The resulting path is shown in Figure 10. The graph did not capture well enough the connectivity of C-Space. However, the interface allows to define new initial and goal configurations and to relaunch the algorithm. The graph was enriched during another 3 minutes, and the maneuver

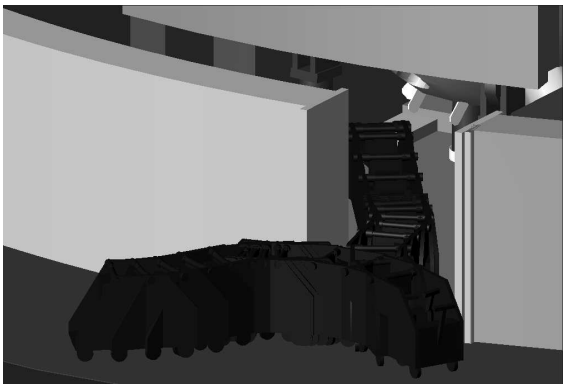

Figure 10: a path.

shown in Figure 11 was found. We repeated this experiment in the original more complex model (model 2), where we used local planner RS during graph generation. In 20 minutes a path was found and a good quality graph of 46 nodes was generated.

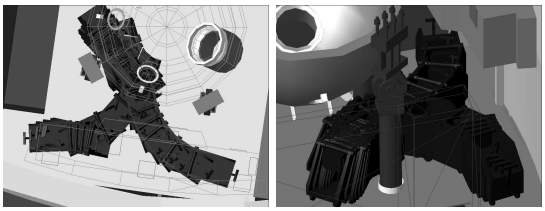

Figure 11: a maneuver.

The table below indicates the geometrical complexity of the respective scenes: the number of solids and the number of facets.

	traveling crane	rotating crane	cart in model 1	cart in model 2
solids	18	270	374	1226
facets	124	8028	17089	27973

Note that the same algorithm was used for all three experiments. Constraints related to the devices were expressed in the choice of the local planner.

The interface allows two ways of use. A learning phase can be launched as a pre-processing step and the resulting graph can then be re-used for several queries. Also, in a more interactive fashion, a graph can be constructed until a particular pair of initial and goal configurations is connected, and extended each time a new query is needed. All experiments were carried out on a SUN Ultra-1, 170 Mhz.

6 Conclusions

In this contribution we described the choices we made to fulfill the needs of a tool for mobility analysis for feasibility studies of maintenance interventions. We described how we implemented a path planner that is well-adapted to the applications, and its interface.

The algorithm is generic in nature and successfully applicable to realistic problems appearing in industrial applications as reported here.

The current level of performance is promising but can certainly be improved. Future developments should include the development of a dedicated dynamical collision checker based upon distance measurement and distance estimation, and applied to the local paths. Also, during the learning phase an automatical decision procedure to switch to another local planner should speed up the convergence of the graph to optimal connectivity.

References

[1] N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspace. In *3rd Workshop on Algorithmic Foundations of Robotics*, Houston, Texas, 5-7 March 1998.

[2] J. Barraquand, L. Kavraki, J.-C. Latombe, T. Li, and P. Raghavan. A random sampling scheme for path plannning. *The International Journal of Robotics Research*, 16(6):759–774, December 1997.

[3] P. Ferbach. A method of progressive constraints for nonholonomic motion planning. In *IEEE Int. Conf. on Robotics and Automation*, 1996.

[4] P. Ferbach and J.-F. Rit. Planning nonholonomic motions for manipulated objects. In *IEEE Int. Conf. on Robotics and Automation*, 1996.

[5] D. Hsu, L. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *3rd Workshop on Algorithmic Foundations of Robotics*, Houston, Texas, 5-7 March 1998.

[6] L. Kavraki, M. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. In *IEEE Int. Conf. on Robotics and Automation*, 1996.

[7] L. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4), 1996.

[8] A. McLean and I. Mazon. Incremental roadmaps and global path planning in evolving industrial environments. In *IEEE Int. Conf. on Robotics and Automation*, 1996.

[9] J. A. Reeds and R. A. Shepp. Optimal paths for a car that goes both forward and backwards. *Pacific Journal of Mathematics*, 145(2), 1990.