A Sampling-Based Tree Planner for Navigation Among Movable Obstacles

Nicola Castaman, Elisa Tosello, Enrico Pagello, Intelligent Autonomous Systems Lab (IAS-Lab), Department of Information Engineering (DEI), University of Padova, Via Gradenigo 6/B, 35131 Padova, Italy
email: nicola.castaman@studenti.unipd.it, toselloe@dei.unipd.it, epv@dei.unipd.it

Abstract

This paper proposes a planner that solves Navigation Among Movable Obstacles problems giving robots the ability to reason about the environment and choose when manipulating obstacles. It finds a path from a robot start configuration $S$ to a goal configuration $G$ taking into consideration the possibility of moving objects if $G$ cannot be reached or if moving objects may significantly shorten the path. The planner combines the A*-Search and the exploration strategy of the Kinodynamic Motion Planning by Interior-Exterior Cell Exploration algorithm. It is locally optimal and independent from the size of the map and from the number, shape, and position of obstacles. It assumes full world knowledge but it can be easily extended in order to explore unknown environments.

1 Introduction

Over the last decades, many Motion Planning algorithms have been developed in order to find a continuous robot motion connecting a robot start configuration $S$ and a goal configuration $G$. Traditional algorithms limit the search of the path within the collision-free space, while avoiding contacts with obstacles in the scene. Moreover, they are not designed for systems with complex dynamics such as humanoids or mobile manipulators.

In a real scenario it could be impossible to find a collision free path connecting two states because of obstructions. Moreover, the minimum cost path allowing a robot to reach a planned pose could not involve a carefully collision-free navigation around the clutter. Sometimes, creating gaps among obstacles can be necessary or can considerably reduce navigation costs. This is what humans do: if it is necessary or less strenuous, humans manipulate objects in order to create free spaces while minimizing efforts and time necessary to reach the goal. Figure 1 depicts a symbolic scenario: a human first tries to move the blue obstacle instead of circumnavigating the collision space that occludes the goal. If the manipulation is too onerous (e.g., the object is heavy, the time required to manipulate the object exceeds the time to perform the collision-free path), then the human will interrupt the task and walk through the existing free space. Future autonomous robots should behave in the same way: service robots, for example, will have to be able to autonomously navigate inside an home or an office while opening doors, moving chairs, etc. Rescue robots should be able to act in areas affected by disasters such as mining accidents, floods, and earthquakes; they should be able to decide when moving obstacles out of the way in order to reach and save human lives as soon as possible. Navigation Among Movable Obstacles (NAMO) gives robots the ability to reason about the environment and choose when manipulating obstacles [1, 2, 3]. It plans robot movements taking into consideration the possibility of moving objects if the goal cannot be reached or if moving objects may significantly shorten the path to the goal.

Figure 1 The Navigation Among Movable Obstacles problem.

Focusing on the structure of actual robots, Motion Planning should not longer ignore their complexity focusing on the resolution of basic geometric problems, such as the piano movers’ problem [4], and hoping on controllers able to follow the paths by keeping velocities slow. It should also account for kinematic and dynamic constraints such as friction, gravity, limits in forces. The need of combining these requirements pushed the research toward techniques with
weak completeness guarantees. Multiple directions of research investigate these techniques. A direction in which promising results have been shown is that of sampling-based motion planning. Sampling-based motion planning algorithms explore the state space of the robotic system by growing a tree of valid motions from the start state of the system towards a goal region, using a model of motion [5]. In detail, complex systems motion planning can be solved by whose tree planners that only depend on forward propagating the model of motion – numerically evaluating motions only forward in time.

The proposed algorithm solves the NAMO problem combining benefits of existing NAMO solvers with that of Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE) [6, 7], a promising sampling-based algorithm. The obtained algorithm addresses a primary NAMO challenge: scalability. It is scalable because of its independence from the size of the map and from the number, shape, and pose of obstacles. It does not impose restrictions on actions to be performed: the robot can both push and grasp every object. It works on a two-dimensional projection of a three-dimensional cluttered workspace letting consider both objects in contact and not in contact with the ground. It assumes full world knowledge but the environment is reconfigurable and the algorithms can be easily extended in order to solve NAMO problems in unknown environments. In fact, it is able to handle sensor feedbacks and correct uncertainties regarding obstacle poses and robot actions. Performed tests prove that it is locally optimal.

The rest of the paper is organized as follows. Section 2 gives a review of related work in NAMO. Section 3 presents the proposed algorithm. Section 4 describes the performed experiments and discusses the obtained results. Conclusions and future works are in Section 5.

2 Related Work

2.1 NAMO

As stated in [2, 3, 8, 9], Wilfong [10] first proved that deterministic NAMO with an unconstrained number of obstacles is NP-hard. Demaine [11] showed that even when considering only unit square obstacles the problem remains NP-hard. In [2], Stilman solved a subclass of NAMO problems, namely $LP_1$, where disconnected components of free-space could be connected independently by moving a single obstacle. These planners solved NAMO problems assuming full world knowledge.

Improvements solve NAMO problems in unknown environments. An example is [8] which discretizes the environment in a $N \times M$ grid and calculates a plan from $S$ to $G$ through an A*-Search using the Euclidean distance as heuristic. Authors presented a baseline as well as an optimized approach. The baseline approach calculates plans for all possible actions on all known objects for any change in the environment. The method does not scale for larger environments. The optimized method does not automatically recompute plans if new information becomes available. When encountering new obstacles, [8] recomputes plans only when the current one becomes invalid due to collisions. For every new obstacle considered for the replan, it evaluates push actions limiting their number by the cost of just avoiding the obstacle itself. It maintains an ordered list of costs relative to the manipulation of each object and, at every step, it selects the minimum cost one. This approach reduces the number of objects that have to be evaluated during the selection but it does not guarantee local optimality, it supports only push actions, and it constrains obstacle shapes to be rectangular. [9] improves [8] guaranteeing local optimality. It supports a larger action set and arbitrary object shapes. It introduces a dynamic bound that limits the number of obstacles evaluations, and it maintains two lists of cost underestimates for every plan. Authors of [9] proved that the algorithm guided the robot to the goal in a time between 18 and 50 seconds in an environment containing between 2 and 70 randomly generated obstacles in randomized configurations.

2.2 Sampling-based Motion Planning

One of the first successful sampling-based motion planners was the Probabilistic Roadmap Method [12]. The algorithm subdivides the search of a valid path into a learning and a query phase. The former takes random samples from the configuration space of the robot, tests them for whether they are in the free space, and uses a local planner in order to attempt to connect these configurations. The latter adds the starting and goal configurations to the graph and applies a graph search algorithm in order to determine a path. Starting from PRM, many other algorithms were developed in order to better guide the tree expansion. Rapidly-exploring Random Trees (RRT) [13] expand from states closed to randomly produced states, Expansive Space Trees (EST) [14] and Single-query Bidirectional probabilistic roadmap planner with Lazy collision checking (SBL) [15] attempt to detect less explored regions and expand from them. KPIECE finally improved the decision phase by making better use of the information collected during the planning process. This information is used to decrease the amount of forward propagation the algorithm needs. As consequence, both runtime and memory requirements decrease making the algorithm suitable to handle high dimensional systems with complex dynamics. As stated in [7], the exploration strategy of KPIECE projects the state space to a lower dimensional Euclidean space and discretizes it by using a grid. The discretization is used to estimate the coverage of the state space and to evaluate cells goodness: for every cell, the algorithm saves the number of times it has been explored and the progress achieved by exploring it. Combining collected information, KPIECE is able to deterministically select the regions to explore: the best less explored ones.

Authors combined existing A*-based NAMO algorithms with the exploration strategy of KPIECE introducing the contributions detailed in Section 1.
3 Proposed Approach

3.1 Problem Statement

Without loss of generality we restrict the domain to a planar projection of the three-dimensional environment. If \( W = \mathbb{R}^n \) is the workspace and \( O \subseteq W \) is the set of obstacles, an instance of the NAMO problem instance can be formally defined by the tuple \( S = (Q, U, S, G, f) \) where:

- \( Q = Q_{\text{free}} \cup Q_{\text{obs}} \) is the robot state space. If \( A(q) \) is the robot in configuration \( q \in Q \), then \( Q_{\text{free}} = \{ q \in Q | A(q) \cap O = \emptyset \} \) is the free space and \( Q_{\text{obs}} = O_{\text{fixed}} \cup O_{\text{movable}} \) is the space populated by obstacles. \( O_{\text{fixed}} \) is the set of polygonal \textit{Fixed Obstacles} that the robot must avoid in navigation and \( O_{\text{movable}} \) is the set of polygonal \textit{Movable Obstacles} that the robot can manipulate by applying forces at allowable contacts;
- \( U \) is the control space;
- \( S \in Q_{\text{free}} \) is the robot start state;
- \( G \in Q_{\text{free}} \) is the goal;
- \( f : Q \times U \to TgQ \) is the forward routine describing the dynamics, where \( TgQ \) is the tangent bundle of \( Q \).

A solution of the NAMO problem instance consists of a sequence of controls \( u_1, ..., u_n \in U \) and times \( t_1, ..., t_n \in \mathbb{R}^{\geq 0} \) such that \( q_0 = S, q_n = G \) and \( q_k \in Q_{\text{free}} \cup O_{\text{movable}}, k = 1, ..., n - 1 \), can be obtained sequentially by integrating \( f \). This means that the motion plan can iterate walking, grasping and moving obstacles until the robot is at \( G \).

3.2 Algorithm

As for KPIECE, every \( q \in Q \) is projected into an Euclidean space \( E \) through a projection \( Proj \). If \( p = Proj(q) \), then the coordinates of \( p \) in \( E \) will be:

\[
Coord(p) = Coord((p_1, ..., p_k)) = ([p_1 - a_1, d_1], ..., [p_k - a_k, d_k])
\]

(1)

where \((p_1, ..., p_k)\) are the components of \( p \), \((a_1, ..., a_k)\) is the origin of the Euclidean map, and \((d_1, ..., d_k)\) is its resolution.

\( E \) is discretized through a grid \( G \) of \( N \times M \) cells of length \( d \). Without loss of generality, \( d \) is chosen as propagation step size of the expansion tree. This means that \( d = d_1 = ... = d_k \) will be the resolution of the map.

\[
Cell(p) = \{ q \in Q | Coord(Proj(q)) \in Cell(p) \}
\]

(2)

defines, for every \( p \in E \), the corresponding cell of \( G \).

A tree data structure \( T \) is defined. Every vertex \( v_i \in T \) refers to the cell \( Cell(i) \in G \); \( v_i \) points to the state of \( Q \) projected into \( Cell(i) \) and used for the propagation. The algorithm proceeds as described in Algorithm 1.

\( T \) is initialized with \( v_S \) referring to \( S \in Cell(S) \). At every iteration, the importance of cells referring to the current node and to its neighbors are updated. The node of the tree referring to the most important cell is selected and a state of it is chosen for the expansion. The process iterates until \( T \) reaches \( Cell(G) \).

The importance of \( Cell(i) \) is defined as:

\[
Importance(i) = \frac{1}{1 + Distance(i) + Weight(i) + Selection(i) + Visits(i)}
\]

(3)

where \( Distance(i) \) is the effort done to cover the Euclidean space.
distance separating Cell(i) from Cell(G); it reflects the A* search. Selection(i) is the number of times that Cell(i) was selected for expansion. Visits(i) refers to the number of times that Cell(i) was considered during the selection phase, namely it is the coverage of Cell(i). Weight(i)$_{TOT}$ is the cost of the path to be performed in order to reach Cell(i), i.e., the sum of the weights of the cells that NAMO sampled as parents of Cell(i). Weight(i) assigned to a cell Cell(i) is defined as follows:

$$\text{Weight}(i) = \alpha \cdot \text{Reach}(i) + \sum_{k} (\beta \cdot \text{Move}(k, i) + \gamma \cdot \text{Return}(i))$$

where Reach(i) is the effort done by the robot in order to reach Cell(i) from the current state; Move(k, i) is the effort required to remove the k-th obstacle from Cell(S) and place it out of the Euclidean distance separating Cell(S) and Cell(G) (0 ≤ k ≤ n, n number of obstacles in Cell(i)). Return(i) is the effort required to come back. In order to homologue data, efforts are represented as time variables.

Importance can be computed in constant time since all the values it depends on can be made readily available. Once visited a cell, its coverage is updated. Once selected a cell from which continuing the expansion, its selection rate is incremented and a robot state within it is sampled. A chain of states Path = (q₀, ..., qᵢ, ..., qₙ) results, with q₀ = S and qₙ = G. The robot real-time performs the Path.

It is easy to observe that, as KPIECE, NAMO prefers expanding from cells that are less covered rather than from cells that are well covered. Cells that have been selected for expansion fewer times are preferred over cells that have been selected many times. Moreover, NAMO gives priority to cells closer to the goal, i.e., less explored areas; and it prioritizes cells that carry the robot to make the least effort, combining navigation and manipulation efforts. Studies show that considering these heuristics in the selection of cells work well in practice [6]. Authors think that formulating these heuristics will facilitate the resolution of the NAMO problem in unknown environments.

4 Implementation Details

4.1 Robot states

An open-source physics simulator is used to approximate the dynamics of the robot. The simulation allows to sample robot states and to define the efforts needed to move obstacles.

4.2 The obstacles relocation process

Once detected the obstacle on the path, the robot has to decide which manipulation action to apply in order to move it. Generally, if the object is small enough (width or length less than the gripper maximum opening), the robot tries to grasp it, otherwise it proceeds with a push. Depending on the action, a different method has been implemented in order to geometrically compute the new object position (the re-estimation of the orientation is unnecessary). Every approach refers to the obstacles positions (current or next) as the positions of their Centers of Mass. Actions are generated for a generic mobile manipulator robot.

4.2.1 The relocation routine of graspable objects

Figure 2 depicts the new positions generation process of a graspable object. Starting from the manipulator origin, n positions are generated around the robot. Every position i (0 ≤ i < n) has the following coordinates:

$$x_i = r_i \cdot \sin \theta_i$$
$$y_i = r_i \cdot \cos \theta_i$$

where

$$r_i = \begin{cases} 
\min & i = 0 \\
\max & i = n - 1 \\
0 & i = 0 \\
\theta_{i-1} + \Delta \theta & \text{otherwise} \\
2\pi & i = n - 1
\end{cases}$$

$$\theta_i = \begin{cases} 
1 & (x_i, y_i) \text{ behind the robot} \\
0.1 & (x_i, y_i) \text{ in front of the robot} \\
-\cos \theta & \text{otherwise}
\end{cases}$$

Δr and Δθ are the angle and radius resolution respectively. min is the minimum reach of the robotics arm allowing the placement of the object out of the footprint polygon of the mobile base and max is its maximum reach.

In order to facilitate the new positions selection, a weight $w_i$ is assigned to every i:

$$w_i = \begin{cases} 
1 & \text{for cell } i \text{ closest to the robot} \\
0.1 & \text{for cell } i \text{ farthest from the robot}
\end{cases}$$

Positions are inserted in an ordered list L depending on $w_i$ and on the distance $d_i$ from the manipulator origin:

$$L = [(x_1, y_1, w_1, d_1), (x_2, y_2, w_2, d_2), ..., (x_n, y_n, w_n, d_n)]$$

with $w_1 > w_2 > ... > w_n$. If $w_i = w_j$, then $d_i \leq d_j$. 
Typically, human navigation routines prefer a forward motion instead of a backward one. Inspired from this behavior, the implemented positions generation process prefers to relocate encountered obstacles behind the robot. This decision should minimize the probability of reconsidering the object again while solving the NAMO problem.

If, during the displacement, no $i$ is kinematically feasible and collision-free, than the constraint $r_i \leq \text{max}$ is relaxed allowing the motion of the mobile base. Depending on the relaxation, $L$ is reformulated.

### 4.2.2 The relocation routine of pushable objects

Figure 3 depicts the positions generation process of a pushable object. A Zig-Zag mode is adopted: new positions are generating to the right or the left of the current one taking into account the space that the robot requires to move. $i$ ($0 \leq i < n$) has coordinates:

$$
\begin{align*}
    x_i &= x_0 + i \Delta d \\
    y_i &= y_0 \pm l
\end{align*}
$$

where $(x_0, y_0)$ is the current position of the object, $\Delta d$ is the distance resolution, and $l$ is the robot maximum side.

As for the Grasp routine, a weight $w_i$ is assigned to every $i$. The adopted rule follows:

$$
    w_i = \begin{cases} 
    1 & \text{if } x_i = x_0 \\
    \frac{1}{d_i} & \text{otherwise}
\end{cases}
$$

where $d_i = D((x_i, y_i), (x_0, y_0))$ is the distance between the new and the current object position with respect to the object reference system. An ordered list

$$
    L = [(x_1, y_1, w_1), (x_2, y_2, w_2), \ldots, (x_n, y_n, w_n)]
$$

is formulated with $w_1 \geq w_2 \geq \ldots \geq w_n$.

In case of failed displacement, the routine increases $l$. $L$ is reformulated.

### 4.2.3 Displacement

The current state of the work requires that objects in the scene are known as well as the gripper poses necessary to grasp or push them. The tuples $(\text{object, gripper_pose})$ are stored in a data set and retrieved when needed.

Once $L$ has been computed, the displacement routine starts: it extracts the positions in $L$ starting from the first and try to place there the object eventually combining navigation and manipulation actions. The routine provides a collision checking of the object during its motion from the current to the goal position.

### 4.3 Sensors feedback

While moving in a workspace, robots may have to deal with unexpected events. Moreover, the pose of the objects in the scene may be subject to uncertainty as well as the motion of the robot. For these reasons, sensors should be mounted on every robot and every automaton should be able to correct its actions based on sensor feedbacks. In our case, the Point Cloud Library [16] has been exploited in order to implement a routine able to read signals of a vision sensor and to process them in order to detect the scene, segment the obstacles, extract their coordinates and eventually recognize them (See Figure 4). The proposed algorithm takes this information in order to update the occupancy map and eventually recompute the NAMO path.

![Figure 4](image_url) Objects in the scene and their segmentations. A marker is visible for every reference system. Objects are ordered depending on the distance from the robot.

### 5 Experiments

In order to evaluate the effectiveness of the proposed solution, two different C++ versions of the algorithm were tested: the proposed one and its random version. As stated in Section 3, the proposed version adds the Euclidean distance (other than the objects weights) to the formulation of the importance, thus combining A* and KPIECE. The random version does not consider the Euclidean distance while evaluating the importance of the neighbors of a cell. It randomly selects a neighbor to be added to the tree. Goals are: - proving the timing improvement achieved by the proposed algorithm; - showing its independence from the size of the map and from the number of obstacles.
Table 1 Mean time of 10000 executions (time expressed in ms).

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>25x25 determ.</th>
<th>25x25 random</th>
<th>50x50 determ.</th>
<th>50x50 random</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>0.265</td>
<td>1.021</td>
<td>0.389</td>
<td>3.742</td>
</tr>
<tr>
<td>100</td>
<td>0.257</td>
<td>1.350</td>
<td>0.380</td>
<td>3.590</td>
</tr>
</tbody>
</table>

Table 2 Mean obstacles weight of 10000 executions.

<table>
<thead>
<tr>
<th>Obstacles</th>
<th>25x25 determ.</th>
<th>25x25 random</th>
<th>50x50 determ.</th>
<th>50x50 random</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>1.056</td>
<td>1.278</td>
<td>1.000</td>
<td>1.333</td>
</tr>
<tr>
<td>100</td>
<td>1.126</td>
<td>1.280</td>
<td>1.000</td>
<td>1.306</td>
</tr>
</tbody>
</table>

All experiments were performed on an Intel Core i7 (2.2GHz) MacBook Pro with 16GB RAM.

5.1 Experimental Setup

Setup 1: Two types of maps were created: one with 70 obstacles and one with 100 obstacles. Every map was discretized into a 25x25 and a 50x50 grid. 100 different maps were generated depending on the size of the grid and on the number of obstacles. For each map, tests were repeated 100 times.

Setup 2: 1) Given a 50x50 grid, the number of obstacles populating it was varied from 10 to 250. 2) Given 70 obstacles, the size of the grid was changed from 20x20 to 100x100.

In both setups, obstacles were polygons of random size, randomly placed on the map. For simplicity, three different weights [1, 3, 5] were considered. Weights were randomly assigned to obstacles.

5.2 Results

Table 1 proves that the elaboration time is not influenced by the number of obstacles. The implemented version is faster than the random one and less affected by an increase of the grid size.

Table 2 shows that, on average, only obstacles with little weight are chosen. The implemented version selects less obstacles than the random one. The choice shows the ability of the robot to select those objects whose displacement requires less effort.

Both proposed and random execution time increases linearly with the growth of the number of obstacles but, as depicted in Figure 5, the random execution time increases faster than the other one. The chart of Figure 6 shows that the random algorithm time increases exponentially with respect to a change of the grid size. On the other side, the implemented algorithm time increases more slowly and linearly. Without loss of generality, these charts prove that the algorithm is independent from the size of the map and the number, weight, size and location of obstacles. In fact, as stated in Subsection 5.1, weights are assigned randomly to obstacles and obstacles are located randomly on the map. Figure 7 depicts a point-like robot performing a path from a Start state to a Goal while adopting the proposed methodology. One obstacle has been selected for the relocation.
Figure 8 The robot in the scene.

5.3 The simulated robot

Once proved the effectiveness of the algorithm for a point-like robot, tests were performed on a simulated one. The implementation of the algorithm has been made ROS-compliant. The open-source Robot Operating System (ROS) [17] is a flexible framework for writing robot software. Its extended use among the robotics community pushed the authors to adopt it as reference frame. The proposed NAMO algorithm extends the ROS navigation package allowing the assignment of different weights to whose cells of the 2D costmap that are populated by obstacles. The navigation of these cells is allowed after the manipulation of inner obstacles. The sole of their 3D shape is remapped into 2D polygons and projected on the grid. The objects manipulation (pushes and grasps) is based on the MoveIt! Simple Grasps tool developed by Dave T. Coleman [18]. This is a simple grasps generator for simple objects for use with the MoveIt! pick and place tool. Tests were performed using an Husky mobile robot with an UR5 robot manipulator, a PR2 gripper, and a Microsoft Kinect vision sensor. Gazebo was used as simulator [19]. Figure 8 depicts a scene of the simulated test. Performance tests are in progress and will be available online at the IAS-LAB GitHub repository.

6 Conclusions and Future Work

The paper presented a NAMO solver combining existing A*-based NAMO algorithms with the exploration strategy of KPIECE. Experiments showed that the algorithm is independent from the size of the map and from the number, shape, and pose of obstacles. Its implementation does not impose restrictions on actions to be performed and is able to handle sensor feedbacks in order to correct uncertainties regarding obstacle poses and robot actions.

A full world knowledge of the workspace is assumed but the algorithm can be easily extended in order to solve NAMO problems in unknown environments. This is one of the future objectives of the authors. When dealing with unknown objects, a predetermined gripper pose is not available. Authors aim to exploit the Reinforcement Learning techniques in order to let robots manipulate unknown objects of any shape. An ontology is being formulating allowing the storage of the information necessary for the manipulation. Its Cloud sharing will speed up the robots ability to manipulate objects thanks to the combined exploitation of their prior knowledge and of the expertise of other robots.

7 Literature


