Tactile-Based Negotiation of Unknown Objects during Navigation in Unstructured Environments with Movable Obstacles

ADVANCED INTELLIGENT SYSTEMS Open Access
www.advintellsyst.com

Tactile-Based Negotiation of Unknown Objects during Navigation in Unstructured Environments with Movable Obstacles

Simon Armleder, Emmanuel Dean-Leon, Florian Bergner, Julio Rogelio Guadarrama Olvera, and Gordon Cheng*

Traditional robot navigation passively plans/replans to avoid any contact with obstacles in the scene. This limits the obtained solutions to the collision-free space and leads to failures if the path to the goal is obstructed. In contrast, humans actively modify their environment by repositioning objects if it assists locomotion. This article aims to bring robots closer to such abilities by providing a framework to detect and clear movable obstacles to continue navigation. The approach leverages a multimodal robot skin that provides both local proximity and tactile feedback regarding physical interactions with the surroundings. This multimodal contact feedback is employed to adapt the robot's behavior when interacting with object surfaces and regulating applied forces. This enables the robot to remove bulky obstacles from its path and solves otherwise infeasible navigation problems. The system's ability is demonstrated in simulation and real-world scenarios involving movable and nonmovable obstacles.

1. Introduction

Seamless navigation of robots within human environments, encompassing homes and offices, is essential for their successful integration into our society. Nevertheless, a significant challenge emerges when robots encounter unfamiliar objects in these environments. Traditional robot navigation techniques primarily rely on a limited search for collision-free paths,[1] following a strategy that prioritizes obstacle avoidance at all costs. Consequently, these algorithms often fail in cluttered situations as they treat all objects as obstacles, neglecting the manipulation capabilities of robotic systems, such as humanoid robots[2,3] or mobile manipulators.[4,5] This approach starkly contrasts with the natural behavior of humans, who leverage their ability to modify the environment in a way that assists locomotion. If necessary, even a child does not hesitate to move an object to reach an obstructed goal.

Inspired by this behavior, our objective is to equip robots with the capacity to analyze the environment and choose when to manipulate an object. Robot movements are planned by considering moving objects when the goal cannot be reached or the path to the goal can be significantly shortened.

Figure 1 shows an example of the type of problem we address in this article. The robot is confronted with reaching a desired goal, obstructed by unknown objects. In such a scenario, a conventional path planner would fail to provide a feasible solution since there is insufficient space to circumvent obstacles. Therefore, the system needs to identify objects, determine whether or not they can be moved, and clear them from the path to create sufficient space for continued navigation. Feedback is provided by the robot's onboard camera, LIDAR sensors on the mobile base, and a robot skin covering its arms and hands.

S. Armleder, F. Bergner, J. R. Guadarrama Olvera, G. Cheng
Institute for Cognitive Systems (ICS)
Technical University of Munich
Germany
E-mail: rogelio.guadarrama@tum.de

E. Dean-Leon
Department of Electrical Engineering, Automation
Chalmers University of Technology
Gothenburg, Sweden

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.202300621.

DOI: 10.1002/aisy.202300621

1.1. State of the Art

Taking into account the possibility of reconfiguring the environment by moving objects is known in the literature as navigation among movable obstacles (NAMO). The first planner that solved NAMO within a practical amount of time was presented in ref. [6]. In this work, the author reduced the search space by assuming that disconnected free-space components can be connected independently by moving only a single obstacle. The strategy was successfully realized on the humanoid Robot HRP-2.[7] However, the authors assumed full knowledge of the environment. Sensor data were acquired through external cameras and markers to position priory known models of movable obstacles. Beyond this, they further solved the domain where maximally k obstacles have to be moved to restore connectivity in the robot's free space, and each object needs to only be moved once.[8]

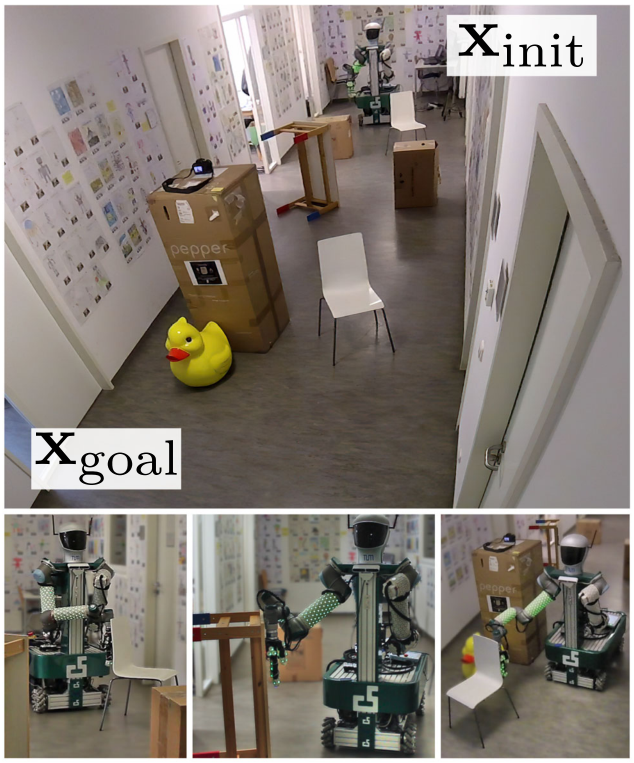


Figure 1. Navigation among movable obstacles: the robot autonomously navigates from the initial pose to the goal pose, negotiating movable and nonmovable obstacles along the way.

All the previously mentioned approaches rely on a complete knowledge of the environment and perfect data. The first extensions of NAMO to unknown environments are presented in refs. [9,10]. The solution in ref. [9] was implemented on the humanoid Robot HRP-2 using onboard sensors only. Movable obstacles are detected by active sensing with a force–torque sensor. Their planning algorithm was a local approach only, where the robot reacts to a specific movable obstacle without considering other obstacles in the computation of a new plan. A planner that achieves local optimality was formulated in ref. [10]. The authors discretize the world into a planar grid and employ A*-Search. The solution is then improved by considering the costs of manipulating or avoiding obstacles.

Only a handful of the proposed algorithms made it to the stage of real-world experimentation.[7,9,11–13] The main challenge is the uncertainty in the perceived world state and the action outcome of a real robotics system. This is why more recent work tries to incorporate these uncertainties directly into the planning algorithm. In refs. [12,14], the authors leveraged ideas from probability theory to formulate the NAMO problem as a Markov Decision Process called NAMO-MDP. In ref. [12], the approach is further extended to incorporate the learned dynamics of movable obstacles. The robot gathers data by applying manipulation forces to one or more points on an object and records the force–torque responses and resulting object trajectory. These data are then used to discover constraints in the object's motion. Scholz et al.[12] evaluated their ideas on a real-world robot but depended on external cameras. They test the algorithm in an environment with two constrained obstacles. In ref. [13], the semantics of movable obstacles are obtained through image scene classification. The system infers movability by assuming that specific object categories are movable. Tests were performed on a mobile manipulator concerning three different kinds of movable obstacles. Wang et al.[15] solved NAMO by extracting object affordances such as pushability and liftability from sensory data. They used trajectory optimization to plan robot movements. The method is showcased in a home environment with a moveable chair and a liftable water bottle. NAMO has also been approached with deep learning techniques. Zeng et al.[16] trained a forward model of object interactions that predicts the outcome of push and pull actions from vision. The method is tested in simulations that usually require removing a single obstacle to reach the goal. Wang et al.[17] trained a reinforcement learning agent and successfully deployed it on a real robot. The method does not include force sensing on the end-effector.

In the literature, knowledge about movability is usually given as input. However, active sensing has also been used based on visual feedback[18] and F/T-sensors.[9,12] Recently, another sensor for physical interactions has become available as whole-body tactile feedback. Skin systems such as refs. [19–21] are inspired by the human sense of touch and have proven to be very useful when acting in unstructured and dynamic environments.[22] Of particular interest is the system developed in ref. [23] that offers multiple modalities such as force, proximity, acceleration, and temperature.

1.2. Our Approach and Contributions

In this article, we address the problem of navigation in a cluttered environment containing reconfigurable objects. Our approach is based on the tactile and proximity feedback measured by a robotic skin system. This sensor allows us to remove the common assumptions about an object's movability and replace them with real-time measurements. For planning object movements, we approximate their geometry with simple boxes and only require some rough information about their position and footprint.

The robot assesses the movability of unknown obstacles by approaching them with its hand and applying a test force. Once it is determined whether or not these obstacles can be moved, a planner decides how to clear the obstacle from the path or, instead, how to navigate around it (Section 2). To address the clearance of large and bulky objects, we employ a compliant pushing strategy that can execute previously planned actions (Section 3). This process repeats iteratively whenever a new obstacle is detected until the robot reaches its goal. Our contribution is a fully functional framework that can solve the navigation among movable obstacles problem in simulation and on a real robot. We validate our approach in cluttered simulated environments containing up to 30 obstacles (Section 4.1) and on a real robot operating in an unknown environment with six different obstacles (Section 4.2).

Overall, our experiments show that the skin on the robot's hand helps to relax the requirements in the perception and planning pipeline. Local information such as object distance, surface shape, and interaction forces can be handled through reactive compliant behaviors rather than exact calibration and planning.

2. Framework

2.1. Overview of Our Approach

The essential steps required by any solution to NAMO problems are: 1) identify the obstacle that requires relocation, 2) maneuver the robot to a suitable position for reaching it with the arm, 3) assess whether the obstacle is movable or static, and 4) negotiate the obstacle by moving it from the path or navigating around it.

These steps have to be solved iteratively until the robot reaches the goal position. The overview in **Figure 2** depicts our approach to the problem.

The information flow begins with an object detector that identifies obstacles in front of the robot. Their position and geometry information is stored in a movability map. Initially, new obstacles are unknown and assumed to be movable. A NAMO planner (Section 2.2) with access to the map simulates possible pushes on detected obstacles to find the one that leads to enough clearance for further navigation. Once completed, the planner returns the next obstacle to move and the desired direction to push. At the next stage, movability detector (Section 3.1.3), the robot approaches the selected obstacle and applies a small test force to determine its movability. The map is updated with this new information. If the obstacle can be moved, the robot initiates a pushing strategy (Section 3.2) to clear the object from its path. Conversely, if the object is deemed static, the planner is reactivated, and with the updated movability map, it has to find another solution. Whenever the robot creates enough free space, the navigation process continues toward the goal. The planner aborts when all known obstacles have been tested, or they are inaccessible, and no viable path can be found.

2.2. NAMO Planner

A plan consists of a navigation sequence and compliant actions designed to bring the robot closer to the goal by systematically removing one movable obstacle at a time from the path. We build upon the planner that was first developed in ref. [10] since it is designed to operate in unknown environments. The algorithm has access to a static 2D map of the environment but no knowledge about obstacles. We make the following assumptions: 1) Connectivity through object manipulation: the planner assumes that two disjoint free space regions can be connected by moving a single object, as illustrated in **Figure 3**. 2) Planar projection of the environment: the planning domain is restricted to a planar projection of the three-dimensional environment. and 3) Obstacle detection and movability assessment: the robot can detect unknown obstacles and actively sense their movability.

The algorithm generates two types of plans: a direct path plan if a clear path to the goal exists without requiring obstacle interaction or a three-step plan if obstacles obstruct the goal. In the latter case, the procedure involves: first, reaching the obstacle (c_1), second, clearing the obstacle (c_2), and finally navigating from the cleared area to the goal (c_3). A visualization of this procedure is shown in Figure 3.

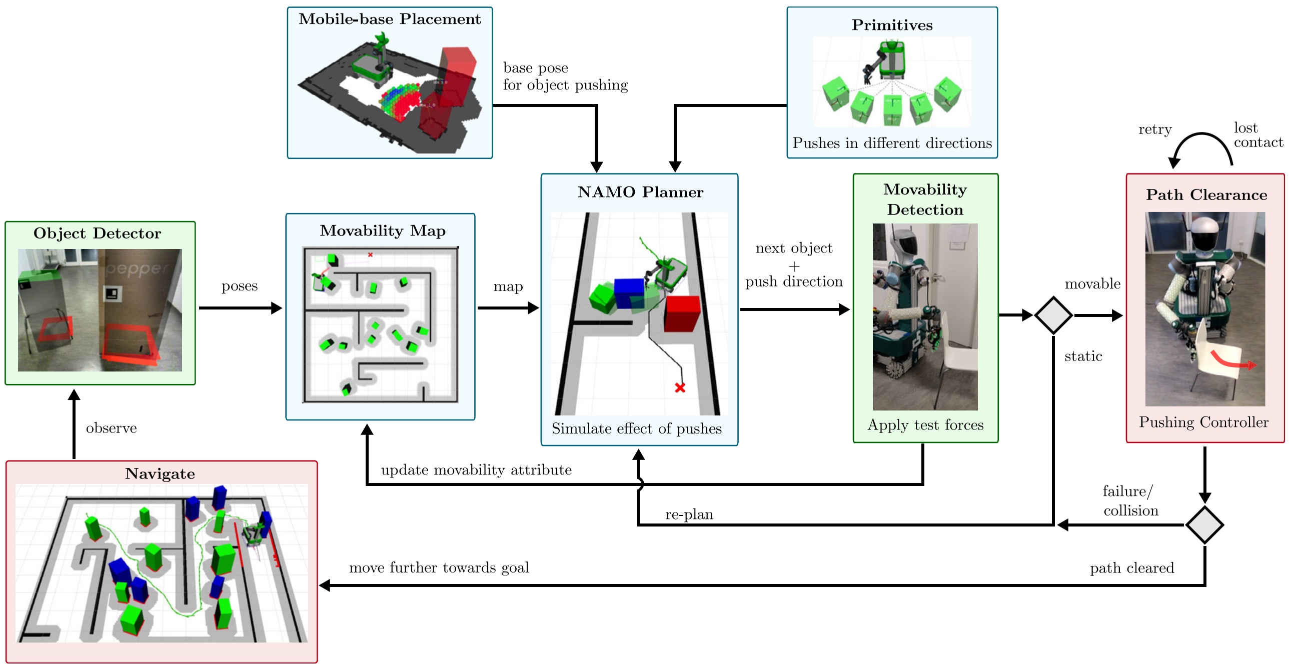


Figure 2. Overview: the goal is to find a sequence of compliant actions that activity creates clearances for navigation. An onboard camera detects new obstacles and adds them to a movability map. Subsequently, a NAMO planner decides which obstacle to move next and in which direction. The robot approaches the selected object and assesses its movability. If it can be moved, a pushing strategy clears it from the path, thereby allowing the robot to proceed further toward the goal. This process repeats iteratively until the goal is reached, or all known obstacles have been tested, and still no path is found.

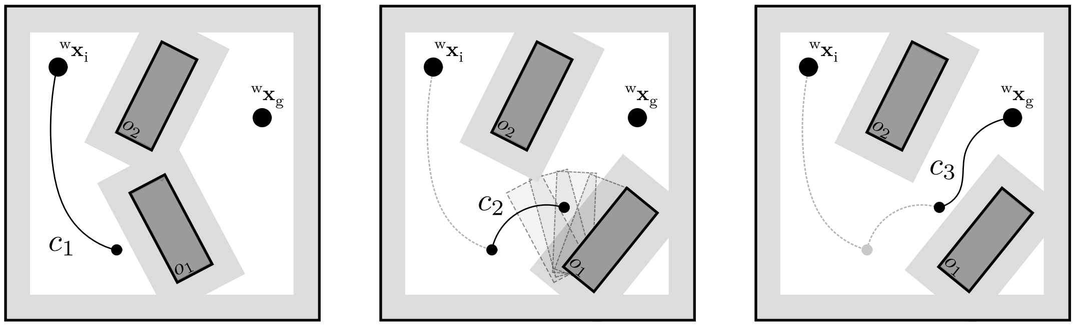


Figure 3. Obstacle plan: a plan between the initial robot pose and the goal pose ${}^{w}\mathbf{x}_g$ involves three segments. Reaching the blocking object (c_1), clearing it (c_2), and finally moving further toward the goal (c_3).

The overall cost of a particular path C_{path} is formulated as a combination of these three segments and based on their Euclidean length:

$$C_{\text{path}}(o, \mathbf{p}, a) = (|c_1| + |c_3|)C_{\text{nav}} + |c_2|C_{\text{act}}(o, \mathbf{p}, a) \tag{1}$$

here, C_{nav} is the step cost associated with moving the robot from one location to another. C_{act} $(o,\ \mathbf{p},\ a)$ denotes the step-cost required to execute clearing action a on object o when making contact at point $\mathbf{p}$ on its surface. The notation $|c_*|$ indicates the Euclidean length of a path segment.

The planning procedure outlined in **Algorithm 1**, finds the best plan according to C_{path}, and can be separated into two steps. First, namoPlan that computes the plan from start to goal $\{{}^{w}\mathbf{x}_i, {}^{w}\mathbf{x}_g\}$ and second obstaclePlan that computes plans for individual obstacles.

2.2.1. NAMO-Plan

Starts by evaluating the direct path between the start ${}^{w}\mathbf{x}_i \in \text{SE}(2)$ and the goal ${}^{w}\mathbf{x}_g \in \text{SE}(2)$ pose, using an A*-Search.[24] This initial path is the current optimal plan, denoted as p^*. The manipulation of objects is only considered as long as it decreases this initial cost. The routine first sorts obstacles based on their distance to the goal and then starts calling obstaclePlan on the current object to see if it can reduce the cost (Line 8 in Algorithm 1).

2.2.2. Obstacle-Plan

Possible plans for a single object are evaluated. This involves simulating the outcome of different clearance actions a on object o and checking for new paths to the goal. The evaluation is performed iteratively, considering different contact points $\mathbf{p} \in \mathbb{R}^2$ on the faces of the object. For each contact point $\mathbf{p}$, a subroutine computes a suitable planar mobile base pose (Section 2.3). If a base pose ${}^{w}\mathbf{x}_b \in \text{SE}(2)$ can be found, the algorithm plans the first path segment c_1 to reach that pose (Line 4). Next, the algorithm iterates over possible clearance actions, applying them to o, and simulating the movements of both the object and the robot. This simulation is performed for a maximum of n_{max} steps. The effort associated with moving the object is determined based on the

Algorithm 1. NAMO-Plan.

```
   Input: $wi$, $wg$, $robot$, $env$
   Output: $p^\star$
1  function namoPlan($wi$, $wg$)
2    $p^\star \leftarrow A^\star wi, wg, env.map$
3    foreach $o \in env.obstacles$ do
4      $o.cost \leftarrow wg - o.\vec{x}$
5    $\mathcal{O} \leftarrow sort(env.obstacles.cost)$
6    $o \leftarrow \mathcal{O}.next$
7    while $o.cost < p^\star.cost$ do
8      $p^\star \leftarrow obstaclePlan(wi, wg, o, p^\star, env)$
9      $o \leftarrow \mathcal{O}.next$
10   return $p^\star$
1  function obstaclePlan ($wi, wg, o, p^\star$, $env$)
2    foreach $\vec{p} \in o.poses$ do
3      $wb \leftarrow robot.\ findBasePose\ (\vec{p}, env.map)$
4      $c_1 \leftarrow A^\star wi, wb, env.map$
5      if $c_1.cost == inf$ then
6        continue
7      foreach $a \in o.actions$ do
8        $env.reset()$, $robot.\vec{x} = wb$
9        $\hat{c} \leftarrow C_{nav} \cdot c_1.cost + wg - robot.\vec{x}$
10       $n_{step}$, $c_2 \leftarrow 0, \varnothing$
11       while $o.movable \wedge \hat{c} < p^\star.cost \wedge n_{step} < n_{max}$ do
12         $o.applyForce\vec{p}$, $robot.doAction(a)$
13         $c_2.addWayPoint(robot.\vec{x})$
14         $c_3 \leftarrow A^\star robot.\vec{x}, wg, env.map$
15         if $c_3.cost < inf$ then
16           $p_o \leftarrow Plan(c_1, c_2, c_3)$
17           if $p_o.cost < p^\star.cost$ then
18             $p^\star \leftarrow p_o$
19           $\hat{c} \leftarrow C_{nav} \cdot c_1.cost + wg - robot.\vec{x}$
20         $env.stepSimulation()$, $n_{step} \leftarrow n_{step} + 1$
21   return $p^\star$
```

displacement of the mobile base *robot*.**x** throughout the interaction and stored as the cost of the second plan component c_2 (Line 13). In each iteration, the algorithm checks for the existence of a new path between the updated robot position and the goal (Line 14). If a valid path component c_3 is found, a complete object plan p_o is formed. If this plan has a lower cost than the current optimal plan p^*, it is updated accordingly (Line 18). There are two termination criteria within the evaluation process. First, if the object o is static or becomes static because it collides with a wall or another static object, the evaluation is stopped by setting *o*.movable to false (Line 11). Second, a termination criterion based on the heuristic $\hat{c}$, which considers the moving costs required to reach the object (c_1.cost) and the estimated remaining distance to the goal $\|{}^{\mathrm{w}}\mathbf{x}_{\mathrm{g}} - robot.\mathbf{x}\|$ (Line 19). This heuristic stops action evaluations as soon as the costs of moving alone become higher than the current optimal p^*, since adding segment c_2 would only increase costs.[10]

After running the planner, it returns the optimal plan, which contains the next navigation goal for the mobile base ${}^{\mathrm{w}}\mathbf{x}_{\mathrm{b}}$, the object to manipulate o, the contact point $\mathbf{p}$ on one of its faces, and the action a to execute.

2.3. Suitable Mobile Base Placement

A mobile-based positioning strategy is necessary to plan path segments for reaching an object before manipulation. Since we are interested in pushing bulky obstacles that might exceed the available workspace of the robot, we have to consider the robot's manipulability.

Different base poses are evaluated against each other by assigning them a performance measure $p(\mathbf{q})$.[25] We consider two factors: arm manipulability $m(\mathbf{q})$ and the distance $d(\mathbf{q})$ of joint angles $\mathbf{q}$ to a preselected arm posture $\mathbf{q}_{\mathrm{post}}$ which is suitable for pushing.

$$\begin{aligned} m(\mathbf{q}) &= \sqrt{\det({}^{0}\mathbf{J}_{\mathrm{h}}(\mathbf{q}){}^{0}\mathbf{J}_{\mathrm{h}}(\mathbf{q})^{\mathrm{T}})} \\ d(\mathbf{q}) &= \textstyle\prod_{i=1}^{n_a}\left[1 - \frac{(q_i - q_{\mathrm{post},i})^2}{\pi^2}\right] \in [0,1] \\ p(\mathbf{q}) &= m(\mathbf{q})(1 + w(d(\mathbf{q}) - 1)) \end{aligned} \tag{2}$$

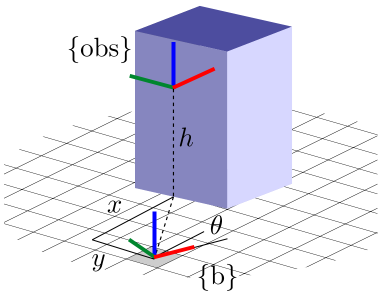
here ${}^{0}\mathbf{J}_{\mathrm{h}}(\mathbf{q})$ is the Jacobian of the hand {h} w.r.t. the robot arm mounting frame {0} and the scalar $w \in [0,1]$ a weight factor, adjusting the influence of the preselected arm posture on the performance index. The goal is to maximize this performance index by selecting a suitable placement for the mobile base relative to a detected object.

If the robot's hand {h} has reached a desired contact point {obs} on one of the object's surfaces, such that ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{h}}(\mathbf{q}) = {}^{\mathrm{w}}\mathbf{T}_{\mathrm{obs}}$. We can re-express this relation using the kinematic chain of a mobile manipulator as:

$$\begin{aligned} {}^{\mathrm{w}}\mathbf{T}_{\mathrm{b}}{}^{\mathrm{b}}\mathbf{T}_{0}{}^{0}\mathbf{T}_{\mathrm{h}}(\mathbf{q}) &= {}^{\mathrm{w}}\mathbf{T}_{\mathrm{obs}} \\ {}^{0}\mathbf{T}_{\mathrm{h}}(\mathbf{q}) &= \left({}^{\mathrm{obs}}\mathbf{T}_{\mathrm{b}}{}^{\mathrm{b}}\mathbf{T}_{0}\right)^{-1} \end{aligned} \tag{3}$$

here ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{b}} \in \mathrm{SE}(3)$ represents the transformation of the mobile base with respect to the world, ${}^{\mathrm{b}}\mathbf{T}_{0} \in \mathrm{SE}(3)$, the fixed transformation between the arm mounting and the mobile base, and ${}^{0}\mathbf{T}_{\mathrm{h}}(\mathbf{q}) \in \mathrm{SE}(3)$, the forward kinematics of the arm. Rearranging for the kinematics of the arm ${}^{0}\mathbf{T}_{\mathrm{h}}(\mathbf{q})$ allows us to determine the joint angles required to reach for an object given the south-after base transformation ${}^{\mathrm{obs}}\mathbf{T}_{\mathrm{b}} \in \mathrm{SE}(3)$. If we assume that the object's geometry can be represented with a simple box during the planning phase, the transformation ${}^{\mathrm{obs}}\mathbf{T}_{\mathrm{b}}$ only depends on rotations around the world z axis, and can be specified with four degrees of freedom:

$${}^{\mathrm{obs}}\mathbf{T}_{\mathrm{b}} = \begin{bmatrix} \mathbf{R}_z(\theta) & \begin{bmatrix} x \\ y \\ -h \end{bmatrix} \\ \mathbf{0}_{3x1}^{\mathrm{T}} & 1 \end{bmatrix}$$



where h is the height of the object and ${}^{\mathrm{obs}}\mathbf{x}_{\mathrm{b}} = [x \quad y \quad \theta]^{\mathrm{T}}$ is the planar pose of the base. By sampling these four parameters within a predefined range and resolution, the corresponding joint solutions and maximum performance index can be obtained through inverse kinematics. Finally, these values are stored in a data structure, indexed by $\{h, x, y, \theta\}$. Building up this data structure is done once in an offline step. During execution, a map $\mathcal{M}$ of possible base positions ${}^{\mathrm{w}}\mathbf{x}_{\mathrm{b}} \in \mathrm{SE}(2)$ for a given ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{obs}}$ is found by slicing the data structure based on the object's height h. The base configuration is then selected by optimizing the performance index within that slice.

$${}^{\mathrm{obs}}\mathbf{x}_{\mathrm{b}}{}^{\star} = \arg\max\nolimits_{{}^{\mathrm{obs}}\mathbf{x}_{\mathrm{b}} \in \mathcal{D}} \mathcal{M} \qquad {}^{\mathrm{w}}\mathbf{x}_{\mathrm{b}}{}^{\star} = {}^{\mathrm{w}}\mathbf{T}_{\mathrm{obs}}{}^{\mathrm{obs}}\mathbf{x}_{\mathrm{b}}{}^{\star} \tag{4}$$

where $\mathcal{D} \subset \mathcal{W}$ is a restricted workspace domain that originates from the required minimum clearances to other obstacles. Examples of generated base poses for pushing are shown in **Figure 4**.

2.4. Internal Simulation of Obstacle Clearance

The planning procedure, described in Algorithm 1, requires the definition of clearance actions capable of removing obstacles from the robot's path. In this work, we consider a restricted set of planar pushes, executed with a constant force, in the directions $\mathcal{A} = \{0, \pm\pi/8, \pm\pi/4, \pm\pi/2\}$rad.

Of course, a broader range of options is available, but we opt for pushing actions as they align closely with the physical capabilities of our real robot. We define possible contact points for each obstacle to apply pushes at the centers of its vertical surfaces.

We leverage a physics engine to simulate the effects of pushes and assess the feasibility of plans before executing them on the physical hardware. The physics engine is called in every iteration of the planner (Line 20) and simulates obstacles' rotation, translation, and collisions when forces are applied. It also takes into account the cascading effect when multiple obstacles collide with

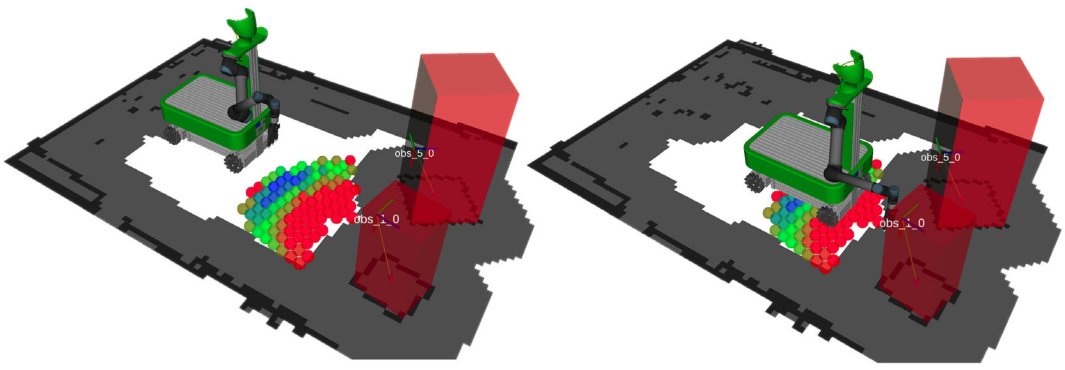


Figure 4. Mobile-base placement: the distribution of possible base poses is visualized as colored spheres on the ground. The blue color indicates a higher performance index.

each other. Given that pushes are inherently planar actions, it is sufficient to simulate movements in two dimensions and exclusively consider the footprint of obstacles. This simplifying assumption increases the efficiency of the simulation step a lot.

An illustrative example of how the planner, the mobile base positioning, and the simulation work together is shown in **Figure 5**. The example considers a hallway scenario with three obstacles, two movable and one declared static. Whenever a new obstacle enters the robot's field of view or an object changes its *movability*, Algorithm 1 is restarted to adapt to the new circumstances.

Of course, such simulations can deviate from reality. Therefore, the following section connects the planner with real-time measurements from the robot and includes strategies for failure detection.

3. Online Execution

3.1. Perception

The essential information required to solve NAMO problems is the movability property of obstacles in the environment. Since we

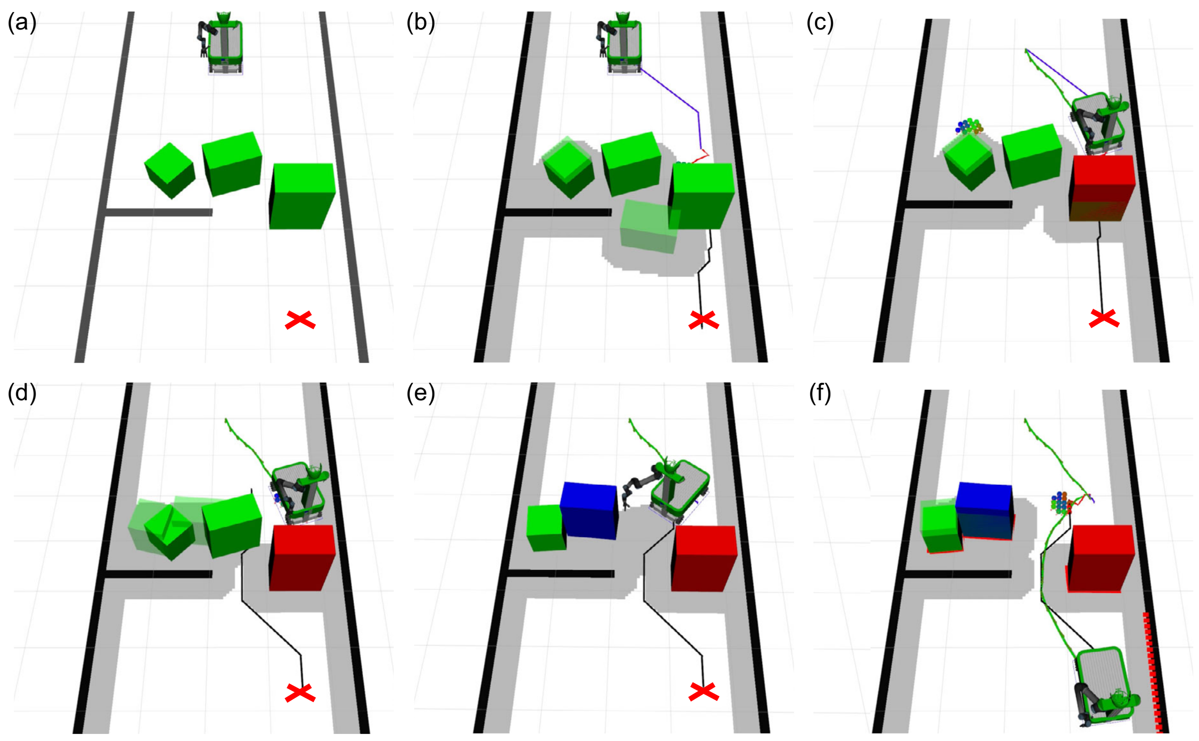


Figure 5. Simulation of NAMO: Initially, all objects are considered movable (colored in green) since their *movability* attribute is unknown. a) The planner determines the lowest cost plan by pushing the right object to the left until there is enough to get to the goal. b) However, when the robot executes this plan, it fails because the object has been declared static (colored in red). c) This triggers a re-planning d). Finally, the updated plan is to push the center object to the left (colored in blue) e). This leads to a valid path to the goal f).

have no prior knowledge about them, the robot needs to employ active sensing.

3.1.1. Robot Skin

One way to gather the information is through a distributed sensing network called robot skin that can cover the surfaces of a robot, see, e.g., Figure 1.

Robot skin obtains real-time measurements about the type, location, and intensity of multiple physical interactions.[23] It is multimodal and senses nearby obstacles through proximity sensors prior to a contact. Forces during physical contact are measured with pressure sensors. Both proximity and force feedback are valuable to realize the required interactions between an obstacle and the real robot: first, to determine object movability, and second, to realize a robust path clearance strategy.

To use the skin's information, its raw cell-wise measurements must be converted into 6D wrenches. Each skin cell inside the sensing network delivers a stream of force $f_i \in \mathbb{R}$ and proximity $p_i \in \mathbb{R}$ information. They are combined into a virtual force $\overline{f}_i = \omega f_i + (1-\omega)p_i$ through a weighing factor ω that allows shifting between force and proximity-based responses. Since the location of every cell is known within the robot's kinematic chain, we can compute the resultant wrench of active cells inside any reference frame, e.g., the hand {h}.

$$^{\mathrm{h}}\mathbf{w} = \begin{bmatrix} ^{\mathrm{h}}\mathbf{f} \\ ^{\mathrm{h}}\boldsymbol{\Gamma} \end{bmatrix} = \begin{bmatrix} \gamma \sum_{i\in\text{active}} \overline{f}_i \, ^{\mathrm{h}}\mathbf{n}_i \\ \gamma \sum_{i\in\text{active}} {}^{\mathrm{h}}\mathbf{t}_i \times \overline{f}_i \, ^{\mathrm{h}}\mathbf{n}_i \end{bmatrix} \tag{5}$$

here $^{\mathrm{h}}\mathbf{n}_i$ and $^{\mathrm{h}}\mathbf{t}_i$ are the normal and translational vector of the i-th cell, and γ scales the signals to treat them as wrenches. Subsequent sections use the obtained skin wrench to implement the robot–obstacle interactions.

3.1.2. Visual Perception for Object Detection

Besides the movability property, the planner requires geometric information about an object. This includes an estimation of its 2D footprint and height. In this work, the primary focus is not on advanced visual perception. Consequently, we achieve object detection through ArUco markers.[26] More advanced object classifiers can replace the simplified approach, as demonstrated in ref. [13].

3.1.3. Tactile Perception for Object Movability

The first interaction between the robot and an obstacle is actively determining its *movability*. This process involves establishing a light contact between the robot's hand and the object's surface. A spline then gradually increases the applied normal force to a predefined maximum, denoted as $f_{\max}$. This maximum force is carefully set to ensure the safety of the robot arm while engaging with the object. Simultaneously, the contact velocity $\dot{x}_{\mathrm{c}} = \mathbf{n}_{\mathrm{c}}^{\mathrm{T}}\mathbf{J}_{\mathrm{c}}\dot{\mathbf{q}}$ is integrated to calculate the displacement Δx_{c} in the object's position. Here, $\mathbf{J}_{\mathrm{c}}$ denotes the contact's Jacobian matrix, $\mathbf{n}_{\mathrm{c}}$ the contact's normal vector, and $\dot{\mathbf{q}}$ the robot's joint velocity vector. If the displacement exceeds a defined threshold distance, denoted by Δx_{th}, the object is classified as *movable*, and the corresponding maximum force required for pushing f_{push} is recorded. **Figure 6** illustrates an example of the discrimination process for a *movable* and *static* object. Note that this method can fail when dealing with soft objects that deform instead of moving linearly. However, more advanced methods might be able to detect soft deformable objects from tactile feedback as well.[27]

3.2. Pushing Things Forward

Once the robot classifies an object as *movable*, an action is required to clear it from the path. We adopt a pushing strategy to deal with large furniture-like objects that can not be grasped.

3.2.1. Pushing Strategy

Our goal is not to precisely position objects in the world but to create enough space for continued navigation. Furthermore, we assume objects behave like holonomic entities with roughly uniform mass distribution.

This allows us to break down the pushing problem into two steps: first, rotating the object toward a desired angle, denoted as θ_{d} and planned by Algorithm 1, followed by a forward push through the object's origin until the path is cleared. **Figure 7**a, b shows a visualization of this procedure. Here, we assign a contact frame $\{c\}$ to the contact point with its x-axis pointing toward the object's origin $\{o\}$. In this new frame, the steps from above can be described as follows: first, a tangential force $\mathbf{f}_t$ is applied in the direction $[0 \quad 1]^{\mathrm{T}}$ primarily causing the object to rotate. Second, a normal force $\mathbf{f}_n$ in the direction $[1 \quad 0]^{\mathrm{T}}$ is applied, which acts through the center of rotation, causing a forward motion.

A proportional controller based on the current orientation error $\delta = \theta_d - \theta$ fuses these two directions and gives the overall pushing direction in the contact frame.

$$^{\mathrm{c}}\mathbf{n} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + k_\delta \delta \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{6}$$

The scalar $k_\delta \in \mathbb{R}$ is a gain parameter used to tune the influence of the orientation error. This strategy leads to trajectories, as visualized in Figure 7c, showing how the object (green) first rotates and then translates while the mobile base (red) follows to maintain contact.

The applied forces have to remain within the object's friction cone to prevent slippage during the pushing. However, determining the exact limits of the cone is difficult. Here, we limit the pushing angle at the object's surface, denoted as $\angle^{\mathrm{o}}\mathbf{n}$, to a fixed range of ± 0.9 rad.

$$\begin{aligned} &^{\mathrm{o}}\mathbf{n} = {}^{\mathrm{o}}\mathbf{R}_{\mathrm{c}}(\alpha)^{\mathrm{c}}\mathbf{n} \\ &\angle^{\mathrm{o}}\tilde{\mathbf{n}} = \min(\max(\angle^{\mathrm{o}}\mathbf{n}, -0.9), 0.9) \\ &\alpha = \operatorname{atan}(t_y, t_x) \end{aligned} \tag{7}$$

By further transforming this direction into the robot's hand frame {h} and multiplying with the desired object speed v_{d}, we get the hand velocity

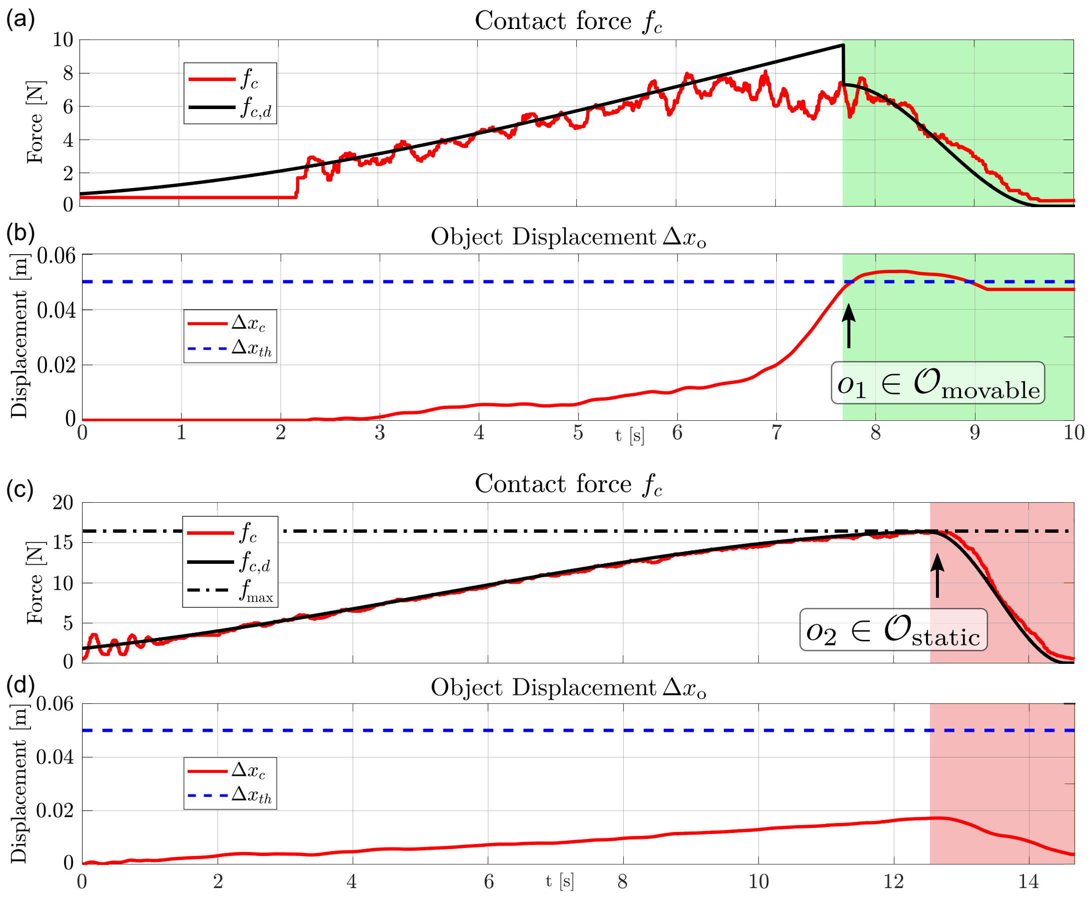


Figure 6. Detection process of movability attribute: object 1 in a), b) is classified as movable after a displacement of $\Delta x_o = 0.05$ m is detected in b). The maximum recorded force before the object starts moving is $f_{\text{push}} = 7$ N. Object 2 in c) d) is classified as static after the maximum effort of $f_{\max} = 18$ N is reached in c). The slight displacement in d) is due to a deformation of the object.

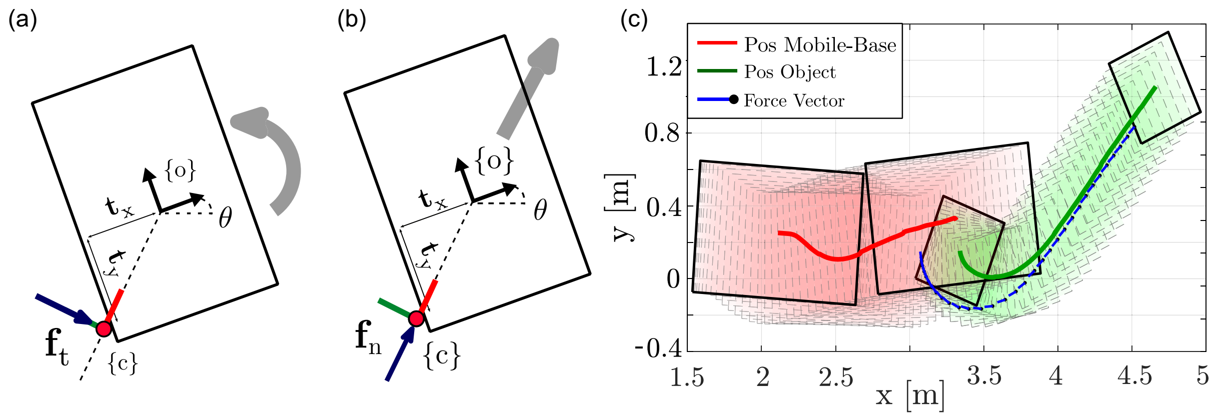


Figure 7. Obstacle pushing: clearing the path is achieved by first rotating a) and then pushing an object forward b) until the path is cleared. This approach leads to the trajectories depicted in c) when pushing at 45° to the left. The mobile base is colored red, and the object is green.

$$ {}^{h}\dot{\mathbf{x}}_d = v_d {}^{h}\mathbf{R}_o \begin{bmatrix} {}^{o}\tilde{\mathbf{n}} \\ 0 \end{bmatrix} \tag{8} $$

We opt for a compliant approach rather than directly sending this command to a velocity controller. Instead, we convert the current velocity error into a desired force that a compliant force controller tracks.

$$ {}^{h}\mathbf{f}_d = \mathbf{K}_v \left({}^{h}\dot{\mathbf{x}}_d - {}^{h}\mathbf{J}_{h,p}(\mathbf{q})\dot{\mathbf{q}} \right) \tag{9} $$

where $\mathbf{K}_\text{v} \in \mathbb{R}^{3\times3}$ is a velocity gain factor and ${}^\text{h}\mathbf{J}_{\text{h,p}}(\mathbf{q}) \in \mathbb{R}^{3\times n}$ is the position part of the hand's Jacobian.

In an unforeseen collision, the controller should not apply more force than a given maximum denoted as f_max. We set f_max slightly higher than the required force for pushing, determined before in Section 3.1.3 when detecting movability.

$$\begin{aligned} \Delta\mathbf{f} &= \gamma\, {}^\text{h}\mathbf{f}_\text{d} - {}^\text{h}\mathbf{f} \\ \gamma &= \min\left(1, \tfrac{f_\text{max}}{\|{}^\text{h}\mathbf{f}_\text{d}\|}\right) \end{aligned} \tag{10}$$

here ${}^\text{h}\mathbf{f}$ is the actual external force sensed at the robot's hand and γ is a factor that limits the applied interaction force to a magnitude of f_max.

The final component of the pushing action involves a strategy to maintain alignment between the hand orientation and the object's surface. This alignment is achieved by following the object orientation reactivity as it turns. The objective can be formulated by ensuring convergence of moments ${}^\text{h}\mathbf{\Gamma}$, measured in the hand frame, to zero

$$\Delta\mathbf{\Gamma} = -{}^\text{h}\mathbf{\Gamma} \tag{11}$$

The complete wrench error $\Delta\mathbf{w} = [\Delta\mathbf{f}^\text{T} \quad \Delta\mathbf{\Gamma}^\text{T}]^\text{T}$ at the hand is then minimized by the whole-body controller in Section 3.3.

3.2.2. Selecting the Contact Point

A remaining question is where to make contact on the object's surface before applying the pushing strategy. As evident from Figure 7a, objects can be turned most easily by exerting forces near the corners. If the goal is to push to the right, the contact point should be located at the left corner and vice versa. The most suitable contact point is in the middle when the objective is to push forward.

The proximity sensing embedded in the skin on the robot's hand offers an elegant solution. By setting the skin wrench in Equation (5) to pure proximity ($w = 0$), the robot can keep a fixed distance to the surface $\mathbf{f}_\text{d} = [0 \quad 0 \quad p_z]^\text{T}$ and scan horizontally along the contour of an object until a jump in the proximity signal indicates the presence of a corner. While searching for the corner, the hand aligns its orientation reactively in the same way as during pushing. Reliably finding the corner of an object through other sensors, such as a camera or force–torque sensor, would be more complex.

3.3. Controller

In the end, all robot actions required in Figure 2, such as reaching for an object, detecting its movability, pushing it, and avoiding collision when the mobile base is in motion, must be converted into torque commands for the real robot.

We make use of our whole-body impedance controller, developed in ref. [28], which combines the holonomic mobile base and the arm into a redundant system $\mathbf{q} = \left[\mathbf{q}_\text{arm}^\text{T} \quad {}^\text{w}\mathbf{x}_\text{b}{}^\text{T}\right]^\text{T}$.

Every action is a combination of low-level compliant controllers, formulated as tasks in Cartesian space[29] for the robot's hand and joint space else. Multiple active tasks are fused in a hierarchical manner using null space projection.[30]

All tasks are expressed in the form of impedance controllers where $\mathbf{\Lambda}, \mathbf{D}, \mathbf{K}$ define the desired inertia, damping, and stiffness matrix. The task error to be minimized is either measured in Cartesian space by $\Delta\mathbf{x} = {}^\text{w}\mathbf{x}_{\text{d,h}} - {}^\text{w}\mathbf{x}_\text{h}$, in wrench space by $\Delta\mathbf{w} = {}^\text{h}\mathbf{w}_\text{d} - {}^\text{h}\mathbf{w}$ or in joint space with $\Delta\mathbf{q} = \mathbf{q}_\text{d} - \mathbf{q}$.

3.3.1. Hand Motion Task

This task tracks a desired trajectory $\{{}^\text{w}\mathbf{x}_\text{d}, {}^\text{w}\dot{\mathbf{x}}_\text{d}, {}^\text{w}\ddot{\mathbf{x}}_\text{d}\}$ and is used for reaching contact points on an object's surface with the robot hand.

$${}^\text{w}\ddot{\mathbf{x}}_\text{h}{}^\star = {}^\text{w}\ddot{\mathbf{x}}_\text{d} + \mathbf{\Lambda}^{-1}({}^\text{w}\mathbf{w}_\text{h} + \mathbf{K}\Delta\mathbf{x} + \mathbf{D}\Delta\dot{\mathbf{x}}) \tag{12}$$

here ${}^\text{w}\mathbf{w}_\text{h}$ is the external wrench measured at the hand to render the motion compliant, especially when the object is detected incorrectly and appears closer than initially anticipated.

3.3.2. Hand Wrench Task

The wrench task controls the force and moment the robot's hand applies to an object. It is used in three ways. First, to determine the movability in Section 3.1.3. Second, to find the corner of an object with the proximity signals. And third, to realize the pushing strategy outlined in Section 3.2.

$${}^\text{h}\ddot{\mathbf{x}}^\star = \mathbf{\Lambda}^{-1}(\mathbf{K}\Delta\mathbf{w} - \mathbf{D}\,{}^\text{h}\dot{\mathbf{x}}) \tag{13}$$

3.3.3. Mobile-Base Collision Avoidance Task

This task prevents the mobile base from colliding with the environment while the hand pushes an object. The method is based on planar artificial potential fields $\mathbf{\Psi}({}^\text{w}\mathbf{x}_\text{b}, \{o\})$ to guide the mobile base away from all close by objects $\{o\}$.[31] The task has lower priority than the other two to not interfere with the hand motions.

$$\tau^\star_\text{oca} = -\mathbf{K}\nabla\mathbf{\Psi}({}^\text{w}\mathbf{x}_\text{b}, \{o\}) - \mathbf{D}\,{}^\text{w}\dot{\mathbf{x}}_\text{b} \tag{14}$$

3.3.4. Arm Posture Task

The lowest priority task is a joint posture task that keeps the arm close to a desired configuration $\mathbf{q}_{\text{d,arm}}$.

$$\boldsymbol{\tau}^\star_\text{post} = \mathbf{K}\Delta\mathbf{q}_\text{arm} - \mathbf{D}\dot{\mathbf{q}}_\text{arm} \tag{15}$$

Finally, in every control cycle, active tasks are evaluated and fused into the next torque command (see ref. [28] for more details).

3.4. Error Handling

Since the real dynamics of an object is unknown, the pushing action is susceptible to errors and requires a recovery strategy in the case of failures. Errors can occur when the object turns unexpectedly, loses contact with the hand, or collides with another static item in the environment. These cases can be detected by monitoring the contact state at the hand to trigger a replaning.

3.4.1. Contact–Loss

A loss of contact is detected in two ways. First, we monitor the sensed normal force component, and if it falls below a predefined minimum threshold, i.e., $f_c < f_{\min}$, this signals a loss of contact. Second, we examine the magnitude of hand moments. If it surpasses a specified maximum threshold, i.e., $\|^{h}\Gamma\| > \mu_{\max}$, there is a misalignment in the hand orientation and the object surface, as per Equation (11). In either case, the pushing action is aborted, and we initiate a second attempt by approaching the object once more.

3.4.2. Collision

Collision scenarios are identified by examining whether the sensed normal force, f_c, exceeds a maximum threshold, $f_{\max}$. In this case, the object is marked as static and no longer considered by the planner in a subsequent replanning step. **Figure 8** shows a collision situation involving a pushed chair and a wall. The objects collide at $t = 8$ s. The compliant pushing controller of Section 3.2 successfully limits the maximum force magnitude to $f_{\max} = 2f_{\text{push}} \approx 9$ N.

4. Experimental Section

We evaluate our approach using two testbeds: 1) a software simulation that can spawn worlds with many obstacles and 2) a real mobile manipulator with a 6 degree of freedom arm.

4.1. Simulation

For the simulation, we use the open-source physics engine Box2d[32] and the ROS Navigation Stack.[33] Box2d efficiently simulates the motion and collision events between many objects. In contrast, the ROS Navigation Stack is a collection of software packages that can guide a robot to a specified goal location while avoiding obstacles.

In our simulation environment, we approximate obstacles with simple boxes of varying sizes. To apply pushing forces, we add potential contact points on each of the four vertical surfaces of these obstacles.

Initially, the robot is supplied with a static map of the environment but does not know the location of obstacles. We assume that new obstacles can be detected once they are within 2 m range of the robot's camera. We set the stepping cost for obstacle clearing in Equation (1) twice as high as the stepping cost for navigation: $C_{\text{act}} = 2.0$ and $C_{\text{nav}} = 1.0$.

To validate the effectiveness of the NAMO planner detailed in Section 2.2, the mobile-base placement in Section 2.3, and the pushing strategy in Section 3.2, we conduct experiments within this simplified setup. As a reference baseline, we use the standard ROS Navigation Stack planner, referred to as *Move_base*. *Move_base* follows the "avoid at all cost" strategy and, therefore, needs to navigate around every single obstacle in the environment.

4.1.1. Random Obstacles

The first experiment aims to address the scalability of the NAMO planner with respect to the number of obstacles in the environment. We configure a 12 m × 12 m maze environment, and the robot's objective is to navigate from a predefined starting pose to a fixed goal pose. Obstacles are added to the map by randomly sampling their positions, orientations, and sizes. Note that there is no guarantee that a direct path to the goal exists.

In each iteration of the experiment, we increase the number of obstacles in steps of 5 s, starting with an empty map and gradually reaching up to 30 obstacles.

We conduct each experiment 10 times, employing both the standard *Move_base* planner and the NAMO planner to collect statistical data. **Figure 9** shows four successive key-frames of the navigation process when the NAMO planner is active, and 25 obstacles are distributed over the map.

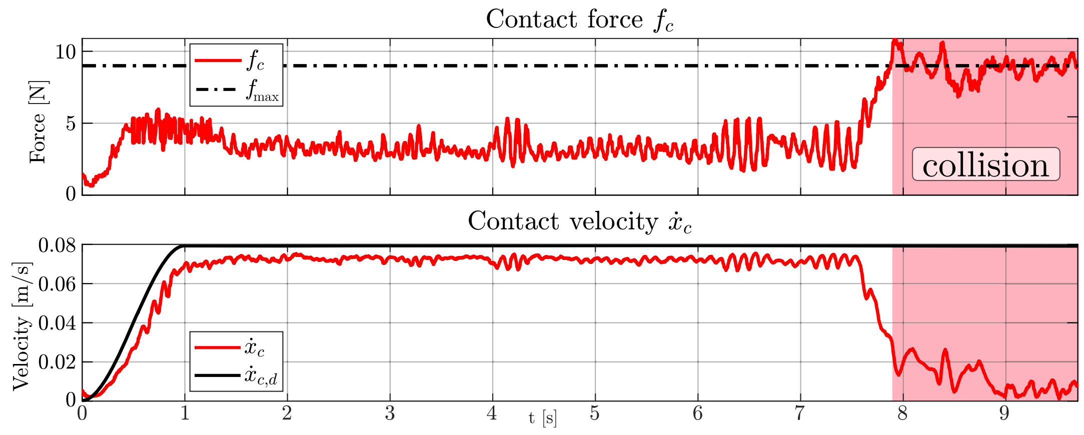


Figure 8. Collision situation: a pushed object accidentally collides with the static environment at 8 s. The compliant pushing controller limits the exerted force f_c to a maximum of $f_{\max} \approx 9$ N a). The corresponding object speed is shown in b).

The experiment's results are depicted in **Figure 10**. In the first plot, we show the total success rate of reaching the goal relative to the number of obstacles in the environment. The *Move_base* planner frequently returns infeasible from 15 obstacles onward when the path to the goal becomes too obscured. The NAMO planner can still reach the target up to the maximum number of obstacles. However, its performance drops when the number of obstacles becomes large because there is often not enough space to position the mobile base when reaching for an obstacle that needs to be moved.

Figure 10b compares the total path length required to reach the goal. In the case of 0–15 obstacles when both planners still manage to complete the task, NAMO consistently produces shorter paths than *Move_base*. *Move_base* often has to take large detours when a new obstacle comes into view and blocks the way.

Quantitative time measurements from the experiment can be found in **Table 1**. It lists the total execution time for NAMO and the time spent in different plan stages relative to the number of obstacles. The occupancy field measures the ratio of occupied map area to total map area after an inflation radius of 0.3 m is applied. The time spent on planning suggests a linear relation between planning time and the number of obstacles.

These simulations show that even if the environment becomes quite cluttered with 30 obstacles (75% of the gird map is occupied space), the common assumption in NAMO[10] of only having to

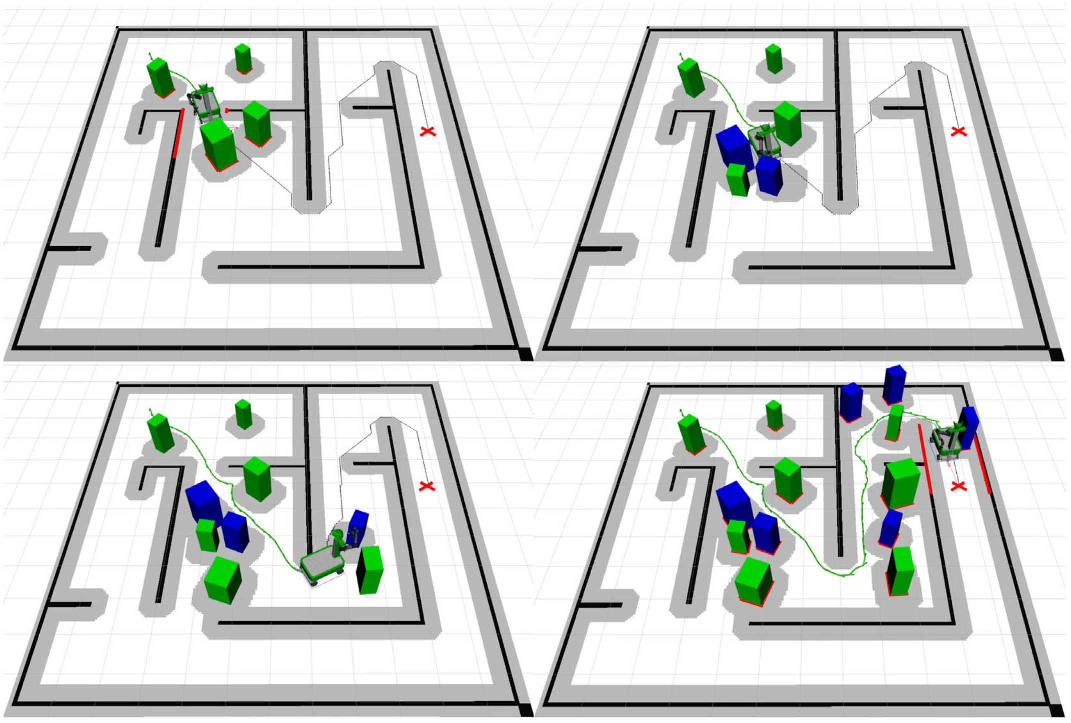


Figure 9. Simulation: four successive key-frames of the random obstacle simulation. The robot discovers objects within its vicinity and pushes them away to reach the goal. Objects that the planner has moved are displayed in blue.

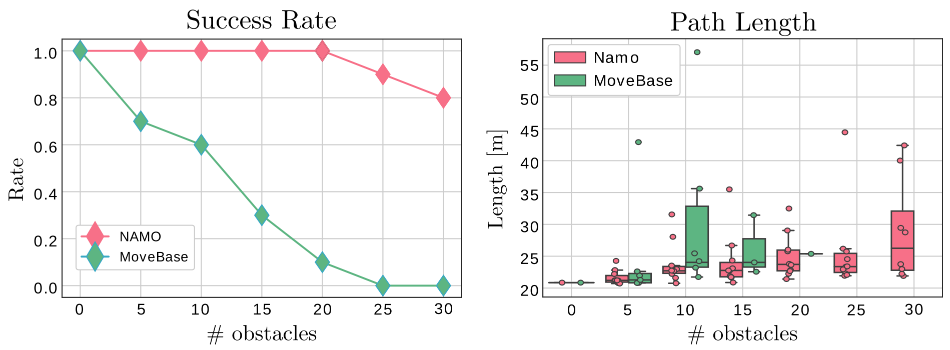


Figure 10. Simulation: comparison between *Move_base* (green) and NAMO (red) in a simulated maze environment. Left: the success rate for reaching the goal. *Move_base* fails to reach the goal when the map contains more than 20 obstacles. Right: the total path length required to reach the goal. Without the ability to move obstacles, the path length is higher.

Table 1. Simulation timings: total execution times for NAMO planner when executed in environments with increasing the number of objects. The planning time scales linearly with the number of obstacles, suggesting that the strategy could be used for even larger maps with more obstacles.

Obstacles	5	10	15	20	25	30
Occupancy [%]	0.48	0.54	0.59	0.64	0.70	0.75
Avg no. of moves	0.6	2.1	3.6	4.2	6.8	9.1
Plan time [s]	1.5	4.1	7.1	14.4	11.3	16.6
Time nav [s]	118.7	150.4	139.9	182.5	169.4	188.0
Assess time [s]	0.3	0.82	1.5	2.8	4.0	7.2
Push time [s]	2.7	3.6	6.7	16.3	20.5	37.0
Total time [s]	123.1	158.8	155.3	216.0	205.2	248.8

move a single object to connect two regions of free space, works well in practice and could potentially scale to even larger maps.

4.2. Real Robot

4.2.1. Experimental Platform

We perform the real experiments on our robotics platform tactile omnidirectional mobile manipulator (TOMM),[4] see Figure 1. TOMM is a dual-arm mobile manipulator composed of two UR5 industrial robots with six degrees of freedom mounted on a holonomic mobile base. In the experiments, we use the right arm on which an Allegro hand is mounted. The robot is equipped with an RGB-D camera in the front and two LIDARs on its base. TOMM's arms and mobile base are velocity-controlled with a fixed control loop frequency of 125 Hz for the arms and 1 kHz for the mobile base. Control algorithms are implemented in C++ using ROS.

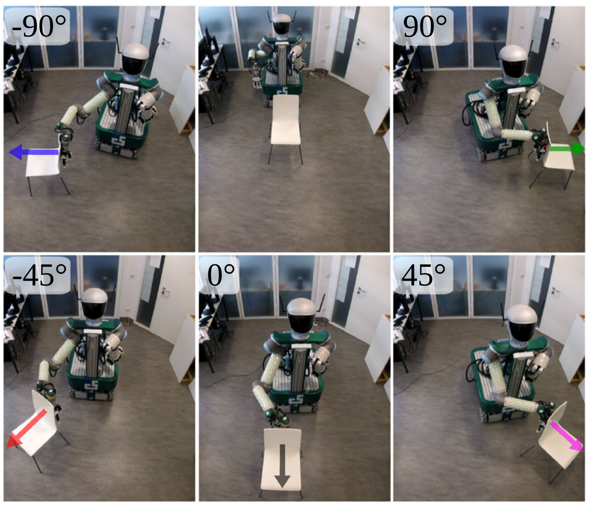
4.2.2. Electronic Skin

For the experiments, we use an electronic skin system (e-skin) capable of large-area tactile sensing.[23] It consists of hexagonal-shaped sensing modules, the *skin cells*.[20] Each skin cell embeds a microcontroller and a set of multimodal tactile sensors. It measures normal forces with three capacitive pressure sensors, light touch and distance with an optical proximity sensor, vibrations with a three-axis accelerometer, and temperature with temperature sensors. Assembled together, these skin cells form bendable *skin patches* mounted on our robot's arms and hands.

Self-configuration and efficient event-driven communication allow the e-skin system to scale to large areas without the need to reduce the effective sample rate of the system.[34] This enables us to deploy 700 cells on the robot that provide information at a frequency of 250 Hz. Manually locating every single cell within the kinematic tree of the robot is impractical. To overcome this challenge, the e-skin can automatically reconstruct its 3D surface coverage and determine the positions and orientations of its sensors. When the proximity signals are converted into distance measures, they can reliably detect objects in the range of 0–0.06 m.

4.2.3. Obstacle Clearance

The initial real-world experiment evaluates the pushing controller as outlined in Section 3.2. In this scenario, the robot is positioned in front of a chair, and the objective is to execute pushes in various directions as specified by the planner in Section 2.4. The chair is marked with an AruCo marker to get a rough estimation of its location through the camera system. Subsequently, the robot approaches this position and comes to rest if the skin proximity sensors on the hand detect the chair's surface. Before pushing can start, the robot needs to locate the object's corner. This is achieved by moving the hand along the surface while maintaining a constant distance with the proximity sensors. The motion



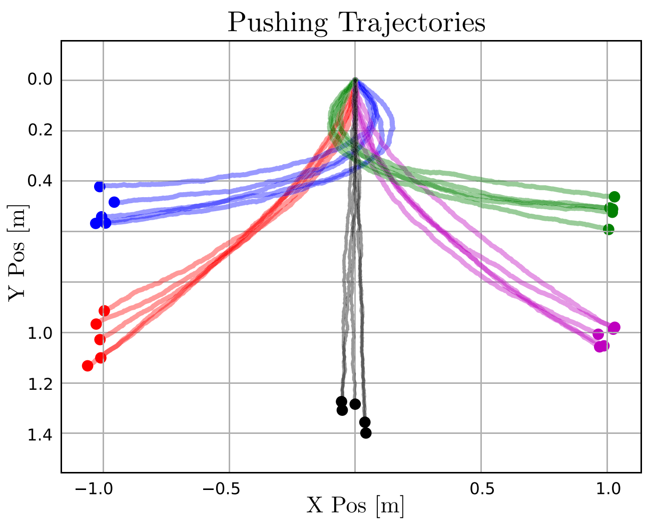


Figure 11. Obstacle clearance: resulting trajectories when clearing obstacles in front of the robot. In this experiment, pushes are executed five times in the directions $\{0, \pm\pi/4, \pm\pi/2\}$rad.

stops if skin cells detect a sudden increase in the distance, indicating the corner location. Finally, the hand makes contact and applies the pushing force based on Section 3.2 to move the chair. We repeat this procedure for the desired pushing directions of $\{0, \pm\pi/4, \pm\pi/2\}$rad, with a path length of 1.4 m. Each direction is repeated five times, and the trajectories are recorded. **Figure 11** shows the resulting motions. The robot can reliably push the obstacle within an area of around 15 cm radius around the goal location. For our intended application, this accuracy is sufficient.

4.2.4. Navigation Among Movable Obstacles

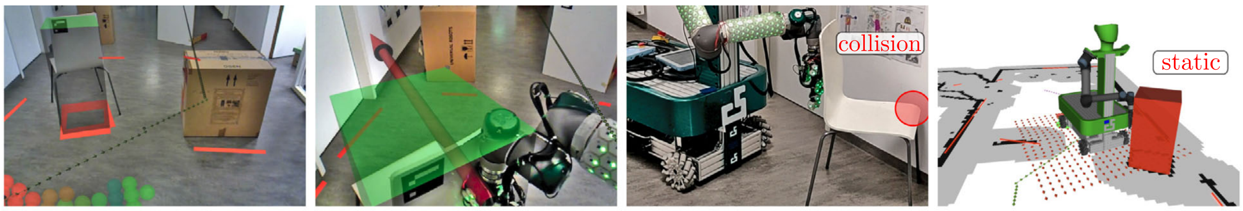
In the final experiment, we tested the entire system within a real-world scenario. The objective is to navigate a distance of approximately 12 m from one end of a hallway to the other. Objects are randomly placed in the environment, and no direct path to the goal position exists, as shown in Figure 1. This scenario highlights the limitations of a standard planner, which would not be able to plot a path.

Like in simulation, we record a static map of the environment without any objects and use it for localization purposes. We selected two chairs, one table, and multiple boxes as obstacles. All of them fulfill the assumptions made earlier in Section 3.2.1 (holonomic entities with roughly isotropic friction).

Some boxes are not labeled with ArUco tags and, therefore, are not visible to the camera system. Nevertheless, they are avoided by the mobile base during navigation and the object-pushing process.

To introduce a static object into the environment, we have increased the weight of the largest box to a level that exceeds the maximum pushing force $f_{\max} = 18$ N of the robot. The system must identify this box as static.

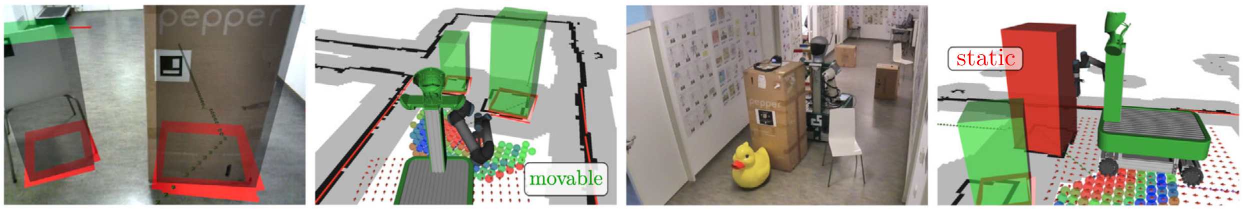
Key-frames of the experiment are shown in **Figure 12**. Planning and execution proceed as follows: the first obstacle that obstructs the path is a chair, as seen in Figure 12a. Once the camera detects it, it triggers a search by the NAMO planner. The planner simulates various pushing actions on the object and selects a leftward push as the most cost-effective action to create free space. The robot then approaches the chair with its hand and

(a)


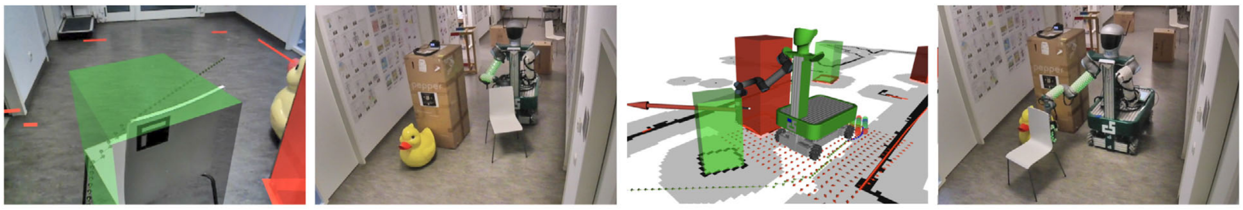
The robot negotiates the first object. While pushing, the object collides with the wall and gets marked as static.

(b)

The second object blocking the path is a table. The plan decides to push the table to the right.

(c)


The robot tries to remove the heavy box. However, the required force is too big. The robot has to find an alternative plan.

(d)


Instead, the robot finds that the chair can be moved and pushes it away to reach the final goal.

Figure 12. Experiment: key-frames of the real execution. The task is to autonomously reach the opposite side of the hallway.

Table 2. Timings real experiment: planning and execution times of the NAMO experiment on the real robot in a 12 m × 3 m corridor. Note that the time for assessing and pushing during real execution also includes the time for unfolding and homing of the robotic arm and is, therefore, much longer than simulation.

	Real	Simulated
Obstacles	**6**	**6**
Occupancy	0.6	0.6
No. of moves	3	3
Plan time [s]	1.6	1.6
Time nav [s]	115.2	104.0
Assess time [s]	119.6	1.4
Push time [s]	135.3	23.1
Force push [N]	{4.6, 8.1, 18.0, 4.3}	–

applies a preliminary test force to assess whether it can be moved. The relatively lightweight chair starts moving at a force of 4.6 N. With this confirmation, the robot executes the push and can continue navigation.

The next visible obstacle blocking the path is the table shown in Figure 12b. Following the same procedure, the planner opts for a push to the right. After checking that it can indeed be moved and locating the table's corner, the robot clears the path.

Then, two new objects appear in the field of view, Figure 12c. The planner attempts to first manipulate the large box since the goal is behind it. However, when applying a test force to the box, it will not move and is therefore classified as static. This triggers a replanning process, leading the robot to consider the chair as the next viable option, Figure 12d. Ultimately, the robot successfully reaches the goal by relocating the chair to the right.

Quantitative results from this experiment are listed in **Table 2**. As before, we show the total time spent in different planning phases and the real force required by the robot to remove obstacles. For comparison, we also replicated the corridor environment in simulation and rerun the experiment. Reality and simulation match the time taken for planning and navigation but differ greatly in accessing movability and pushing. This discrepancy arises because the movability attribute in simulation is determined instantaneously and because the real robot needs to perform additional actions, such as folding and unfolding its arms.

5. Conclusion

In this work, we tackled the challenge of enabling robots to navigate common human environments filled with unknown and potentially movable obstacles. The framework we developed effectively addressed this problem and allowed our robot to reach desired goals even if the path was obscured.

Instead of relying on given knowledge about an obstacle's manipulability, we use tactile signals from a robot skin system to actively sense movability.

Object movements are planned in a simulator before executing them on the real robot and only require approximate information about the object's location and a 2D footprint.

To remove blocking obstacles from the robot's path, we developed a pushing strategy that can realize the planner's action on a real robot. The method employs proximity information to reach for unknown obstacles and detect their corner location. Then, a feedback controller based on tactile information pushes the obstacle aside, creating space for further navigation. During pushing, the system can identify collisions with other static items of the environment and unexpected object motions to trigger a second attempt. Our simulation experiments show that the NAMO approach substantially increases the chance of success over standard "avoid at all costs" navigation. In simulated environments where ≈75% of the map is occupied, NAMO was still able to reach the goal in 80% of the cases.

Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Keywords

e-skin, mobile manipulation, motion planning, navigation among movable obstacles, tactile interactions

Received: September 30, 2023
Revised: November 7, 2023
Published online:

[1] J.-C. Latombe, *Robot Motion Planning*, The Springer International Series in Engineering and Computer Science (SECS), Vol. *124*, Springer Science & Business Media, New York **2012**.

[2] G. Cheng, S.-H. Hyon, J. Morimoto, A. Ude, J. G. Hale, G. Colvin, W. Scroggin, S. C. Jacobsen, *Adv. Rob.* **2007**, *21*, 1097.

[3] K. Kaneko, K. Harada, F. Kanehiro, G. Miyamori, K. Akachi, in *2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, IEEE **2008**, pp. 2471–2478.

[4] E. Dean-Leon, B. Pierce, F. Bergner, P. Mittendorfer, K. Ramirez-Amaro, W. Burger, G. Cheng, in *2017 IEEE Int. Conf. on Robotics and Automation (ICRA)*, IEEE **2017**, pp. 2441–2447.

[5] T. Asfour, L. Kaul, M. Wächter, S. Ottenhaus, P. Weiner, S. Rader, R. Grimm, Y. Zhou, M. Grotz, F. Paus, D. Shingarey, H. Haubert, in *2018 IEEE-RAS 18th Int. Conf. on Humanoid Robots (Humanoids)*, IEEE **2018**, pp. 447–454.

[6] M. Stilman, J. J. Kuffner, *Int. J. Humanoid Rob.* **2005**, *2*, 479.

[7] M. Stilman, K. Nishiwaki, S. Kagami, J. J. Kuffner, in *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* **2006**, pp. 820–826, ISSN: 2153-0866.

[8] M. Stilman, J. Kuffner, *Int. J. Rob. Res.* **2008**, *27*, 1295.

[9] Y. Kakiuchi, R. Ueda, K. Kobayashi, K. Okada, M. Inaba, in *2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* **2010**, pp. 1696–1701, ISSN: 2153-0858.
[10] H.-N. Wu, M. Levihn, M. Stilman, in *2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* **2010**, pp. 1433–1438, ISSN: 2153-0858.
[11] M. Levihn, L. P. Kaelbling, T. Lozano-Pérez, M. Stilman, in *2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* **2013**, pp. 224–231, ISSN: 2153-0866.
[12] J. Scholz, N. Jindal, M. Levihn, C. L. Isbell, H. I. Christensen, in *2016 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, IEEE **2016**, pp. 3706–3713.
[13] Z. Meng, H. Sun, K. B. H. Teo, M. H. Ang, in *2018 IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics (AIM)* **2018**, pp. 156–163, ISSN: 2159-6255.
[14] M. Levihn, J. Scholz, M. Stilman, in *Algorithmic Foundations of Robotics X*, Springer Tracts in Advanced Robotics (Eds: E. Frazzoli, T. Lozano-Perez, N. Roy, D. Rus), Springer, Berlin, Heidelberg **2013**, pp. 19–35, ISBN 978-3-642-36279-8.
[15] M. Wang, R. Luo, A. Ö. Önol, T. Padir, in *2020 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, IEEE **2020**, pp. 2734–2740.
[16] K.-H. Zeng, L. Weihs, A. Farhadi, R. Mottaghi, in *2021 IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, IEEE **2021**, pp. 9863–9872.
[17] H.-C. Wang, S.-C. Huang, P.-J. Huang, K.-L. Wang, Y.-C. Teng, Y.-T. Ko, D. Jeon, I.-C. Wu, *IEEE Rob. Autom. Lett.* **2023**, *8*, 2740.
[18] H. Sun, Z. Meng, M. H. Ang, in *2017 IEEE Int. Conf. on Cybernetics and Intelligent Systems (CIS) and IEEE Conf. on Robotics, Automation and Mechatronics (RAM)* **2017**, pp. 207–212, ISSN: 2326-8239.
[19] G. Cannata, M. Maggiali, G. Metta, G. Sandini, in *2008 IEEE Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems*, IEEE **2008**, pp. 434–438.
[20] P. Mittendorfer, G. Cheng, *IEEE Trans. Rob.* **2011**, *27*, 401.
[21] A. Cirillo, F. Ficuciello, C. Natale, S. Pirozzi, L. Villani, *IEEE Rob. Autom. Lett.* **2015**, *1*, 41.
[22] A. Jain, M. D. Killpack, A. Edsinger, C. C. Kemp, *Int. J. Rob. Res.* **2013**, *32*, 458.
[23] G. Cheng, E. Dean-Leon, F. Bergner, J. R. G. Olvera, Q. Leboutet, P. Mittendorfer, *Proc. IEEE* **2019**, *107*, 2034.
[24] P. Hart, N. Nilsson, B. Raphael, *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100.
[25] N. Vahrenkamp, T. Asfour, R. Dillmann, in *2013 IEEE Int. Conf. on Robotics and Automation*, IEEE **2013**, pp. 1970–1975.
[26] F. J. Romero-Ramirez, R. Muñoz-Salinas, R. Medina-Carnicer, *Image Vision Comput.* **2018**, *76*, 38.
[27] M. Kaboli, G. Cheng, *IEEE Trans. Rob.* **2018**, *34*, 985.
[28] S. Armleder, E. Dean-Leon, F. Bergner, G. Cheng, *Adv. Intell. Syst.* **2022**, *4*, 2100047.
[29] O. Khatib, *IEEE J. Rob. Autom.* **1987**, *3*, 43.
[30] A. Dietrich, C. Ott, A. Albu-Schäffer, *Int. J. Rob. Res.* **2015**, *34*, 1385.
[31] O. Khatib, *Int. J. Rob. Res.* **1986**, *5*, 90.
[32] E. Catto, https://github.com/erincatto/box2d (accessed: December 2020).
[33] K. Zheng, *Robot Operating System (ROS): The Complete Reference (Volume 6)* (Ed: A. Koubaa), Studies in Computational Intelligence 962, Vol. *6*, Springer, Cham, Switzerland **2021**, pp. 197–226.
[34] F. Bergner, E. Dean-Leon, G. Cheng, *Sensors* **2020**, *20*, 1965.