Pushing Through Clutter with Movability Awareness of Blocking Obstacles

Weeda, Joris J.; Bakker, Saray; Chen, G.; Alonso-Mora, Javier

DOI
10.1109/ICRA55743.2025.11127788

Publication date
2025

Document Version
Final published version

Published in
Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2025

2025 IEEE International Conference on Robotics and Automation (ICRA)
May 19-23, 2025. Atlanta, USA

Pushing Through Clutter With Movability Awareness of Blocking Obstacles

Joris J. Weeda, Saray Bakker, Gang Chen and Javier Alonso-Mora

***Abstract*— Navigation Among Movable Obstacles (NAMO) poses a challenge for traditional path-planning methods when obstacles block the path, requiring push actions to reach the goal. We propose a framework that enables movability-aware planning to overcome this challenge without relying on explicit obstacle placement. Our framework integrates a global Semantic Visibility Graph and a local Model Predictive Path Integral (SVG-MPPI) approach to efficiently sample rollouts, taking into account the continuous range of obstacle movability. A physics engine is adopted to simulate the interaction result of the rollouts with the environment, and generate trajectories that minimize contact force. In qualitative and quantitative experiments, SVG-MPPI outperforms the existing paradigm that uses only binary movability for planning, achieving higher success rates with reduced cumulative contact forces. Our code is available at: https://github.com/tud-amr/SVG-MPPI**

***Index Terms*— Motion and Path Planning, Collision Avoidance, Integrated Planning and Control**

I. INTRODUCTION

A fundamental ability of autonomous robots is to navigate towards a goal while avoiding collisions along the way [1]. However, in complex and cluttered environments, such as domestic settings where obstacles like chairs and boxes may obstruct the path to the goal, finding collision-free paths often becomes impractical. In such cases, traditional navigation methods often fail and Navigation Amongst Movable Obstacles (NAMO) becomes essential.

NAMO involves actively interacting with the environment by relocating obstacles to clear a path, enabling successful navigation to the target. The concept was initially explored by Reif and Sharir [2], and later formally introduced as a category of research by Stilman and Kuffner [3]. These works demonstrated that solving NAMO in a two-dimensional workspace with movable obstacles is NP-hard, and NP-complete in simplified cases with unit-square obstacles, due to the large search space and the numerous possible configurations [2, 4]. As a result, most solutions are approximations rather than exact solutions, and practical applications remain limited [5]. Research often focuses on simplified versions of the problem, such as LP_1, a linear program where a single obstacle blocks the path, or LP_2, which involves two obstacles [3, 6].

This project has received funding from the European Union through ERC, INTERACT, under Grant 101041863. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

The authors are with the Cognitive Robotics Department, TU Delft, 2628 CD Delft, The Netherlands {jjweeda, s.bakker-7, g.chen-5}@tudelft.nl

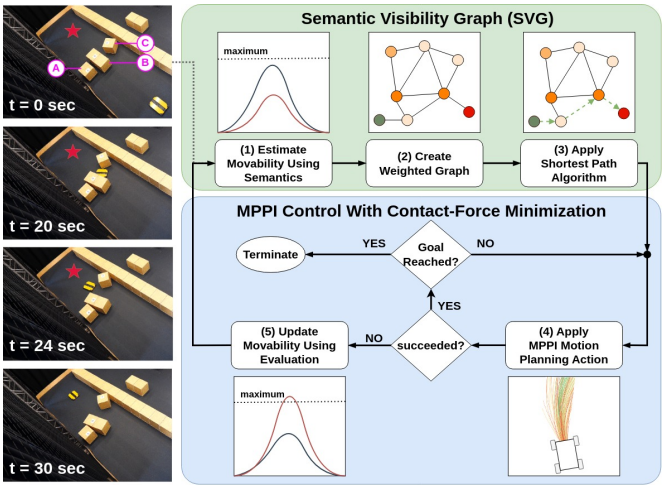


Fig. 1: An overview of the proposed SVG-MPPI architecture where the SVG provides a weighted graph with efficient node placement around movable obstacles along which a lowest-effort path can be found. The generated set of waypoints guides the MPPI control strategy to efficiently sample rollouts around movable obstacles. If during interaction an obstacle is considered non-movable, the movability estimation gets updated and the path is replanned. Snapshots of a real-world example are shown on the left where the red star indicates the goal location and the masses of the obstacles are (A): 25 kg, (B): 20 kg, (C): 5 kg.

Existing NAMO approaches [3–5, 7–12] and sequential nonprehensile manipulation strategies [13, 14] consider only binary movability information, i.e., whether obstacles are movable or not, neglecting the varying levels of effort required to move different obstacles and thus lacking efficiency. Furthermore, these works rely on task planners to decide when the robot should move without displacing an obstacle (transit) or when it should displace an obstacle (transfer) and where to place the obstacle. However, obstacle placement planning itself is a difficult problem and these works typically function only in simplified scenarios, such as the LP_1 and LP_2 scenarios. How to realize efficient NAMO in more cluttered and random environments remains an open problem.

This work incorporates continuous movability information into the NAMO-solving process, improving efficient navigation around obstacles with contact force minimization while moving to the goal. Moreover, inspired by the framework in [5], we use a physics engine to model state transitions of the obstacles in the environment. We introduce a framework that can rapidly simulate the interaction outcomes of each rollout first before selecting the best one, without the need for explicit obstacle placement planning, in contrast to [5]. These rollouts are planned using an SVG-MPPI approach, which tackles global planning with movability-aware node

2025 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3315-4139-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICRA55743.2025.11127788

placement and local planning with the consideration of contact force minimization using a Model Predictive Path Integral (MPPI) control strategy. Fig. 1 shows the framework of our system and snapshots of a real-world experiment.

The main contributions of this work are:

- *Integration of Continuous Quantified Movability:* We consider obstacle movability on a continuous scale rather than as a binary property, leading to more efficient planning in cluttered environments.
- *Contact-Force Minimization in MPPI control:* Our approach minimizes contact forces between the robot and the environment using the movability data and a physics engine, reducing the effort required to move objects.
- *Elimination of Explicit Obstacle Placement:* We eliminate the requirement for explicit obstacle relocation, enabling applicability in more cluttered random environmental setups.

II. PRELIMINARIES

This section presents the key concepts underlying our approach. We begin by defining the necessary notations, followed by an overview of the Visibility Graph (VG) method which provides the basis for the SVG. Lastly, we describe the MPPI control strategy, which is integrated with a physics engine to model obstacle interactions.

A. Notation

In this paper, a graph is represented as $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes with n the number of nodes, and E the edges. Each obstacle is described as a polygon O_i, with mass m_i. The distance between a node v_i and an obstacle is denoted as d_i, and the robot's safety margin is given by r. *Free-space nodes*, V_{free}, originate from the visibility graph and do not require manipulation, while *passage nodes*, V_{passage}, involve obstacle manipulation. The robot's state is denoted $\mathbf{x}_t$, its velocity $\dot{\mathbf{x}}_t$, and control inputs are given by $\mathbf{v}_t$. A set of waypoints is represented by $\mathcal{P} = \{\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p}_{n_p}\}$ with n_p the number of positions $\mathbf{p}$.

B. Visibility Graphs (VG)

VG is a key tool in path planning, representing direct line-of-sight connections between significant points in an environment [15]. These connections, or edges, create a network that allows algorithms such as Dijkstra's [16] or A* [17] to compute optimal paths between start and goal nodes. VG construction involves identifying key points, typically at obstacle corners, and connecting them with edges wherever visibility is unobstructed. To ensure safe navigation, obstacle boundaries are inflated by a margin r, resulting in a graph $G = (V_{\text{free}}, E)$ that facilitates efficient path planning, particularly when obstacles are static. Fig. 2a demonstrates a basic visibility graph applied to a straightforward environment, with inflated obstacles and points placed in the remaining free space. However, Fig. 2a also highlights a limitation of VGs: when the goal is blocked by movable obstacles, certain locations are not mutually visible, creating gaps in the visibility graph. Consequently, VG alone becomes insufficient for generating a path to the goal in such scenarios.

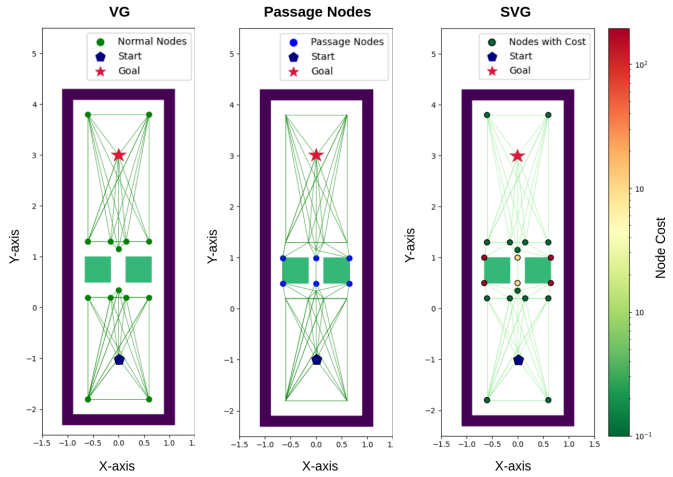


Fig. 2: Comparison of different visibility graph strategies: (a) The normal Visibility Graph (VG), (b) A VG enhanced with Passage Nodes, and (c) A Semantic Visibility Graph (SVG) where node costs reflect obstacle manipulation effort.

C. Model Predictive Path Integral (MPPI)

MPPI is a sampling-based control method used for navigating dynamic environments by generating multiple control input sequences and evaluating them using a cost function [18]. MPPI involves sampling a total of K input sequences V_k that generate the state trajectories Q_k, $\forall k \in K$, often referred to as rollouts. Each rollout Q_k represents a series of state vectors $\mathbf{x}_{t,k}$, meaning that each rollout k consists of prediction steps t over a total horizon of T. Using a cost function C_k, each rollout is evaluated where rollouts that violate the constraints are disregarded. A weighted sum of all rollouts determines the next action for the robot. In the work by Pezzato et al., MPPI is integrated with IsaacGym [20], a physics engine that simulates the robot's environment in parallel using GPU power, allowing for rapid computation of multiple trajectories and the elimination of an explicit model of obstacle interactions [19].

III. METHOD

This section introduces SVG-MPPI, a framework for navigating cluttered environments without explicit object placement. It combines the Semantic Visibility Graph (SVG) for path planning and MPPI with contact force minimization for robot control (Fig. 1). Sec. III-A details the SVG algorithm, which adds passage nodes to connect disconnected regions. Sec. III-B explains the MPPI control strategy, using real-time feedback from IsaacGym to minimize contact forces. Finally, Sec. III-C covers movability evaluation for online replanning based on obstacle interactions.

A. Semantic Visibility Graph (SVG)

The SVG extends the traditional VG by incorporating continuous movability, allowing the placement of *passage nodes* near movable obstacles while accounting for the push actions required to access them. These nodes connect previously disconnected areas, creating new pathways for the

robot, as shown in Fig. 2b-c. In the initial phase (lines 1-11 of Algorithm 1), the process follows the conventional VG approach [15]. In the second phase (lines 12-19 of Algorithm 1), passage nodes are added near movable obstacles where the space between pairs of obstacles is too narrow for the robot to pass. These passage nodes are integrated into the weighted visibility graph like any other node. However, accessing these nodes requires the robot to manipulate the obstacle as the safe distance margin is breached.

Movability is treated as a continuous variable, based on the mass m_i of each obstacle. Objects with masses below a threshold are considered movable, with lighter objects preferred for manipulation. Movability, defined by mass, can be estimated using perception-based machine learning models such as Image2Mass [21], or derived from obstacle interactions [22–25]. However, the process of deriving mass from environmental data is beyond the scope of this work, meaning mass distributions are assumed to be known.

Passage nodes are created by identifying candidate obstacle pairs. Two obstacles, O_i and O_j, form a pair if they are close together, at least one is movable, and the gap between them is too narrow for the robot to pass. The boundaries are approximated using convex hulls to ensure accurate node placement for both convex and non-convex shapes. Passage nodes are then created through a three-stage process, as visualized with two examples in Fig. 3. In Stage I, the nearest points on the edges of the objects are identified that lie on the line-segments connecting the vertices of O_i to the other object O_j, to estimate the passage area. By selecting the closest points within this set, an area is created for connecting passage nodes to each other and the disconnected regions. In Stage II, entry and exit boundaries are drawn from the convex hull of all the nearest points, defining the line segments where passage nodes will be placed. Finally, In Stage III, passage nodes V_{passage} are placed along entry and exit boundaries, treating them interchangeably in an undirected graph. Each passage node is repositioned along

Algorithm 1 Semantic Visibility Graph

Input: A set O of disjoint polygonal obstacles.
Output: The visibility graph $G(V, E)$.
Initialize $V \leftarrow \emptyset$; $E \leftarrow \emptyset$
for $i \leftarrow 0$ to $|O| - 1$ **do**
  Compute $P_i \leftarrow$ vertices of inflated polygon O_i with safety margin r
  for $j \leftarrow 0$ to $|P_i| - 1$ **do**
    Select $v \leftarrow P_i[j]$
    Compute $W \leftarrow \text{visibleVertices}(v, O)$
    for each vertex $w \in W$ **do**
      Update $E \leftarrow E \cup \{(v, w)\}$
      Update $V \leftarrow V \cup \{v\}$
for each pair (P_i, P_j) where $i \neq j$ **do**
  if distance between P_i and P_j is less than $2r$ **then**
    if either $\text{mass}(P_i) \leq$ max mass or $\text{mass}(P_j) \leq$ max mass **then**
      Compute $p \leftarrow \text{passageNodes}(P_i, P_j)$
      Compute $W \leftarrow \text{visibleVertices}(p, O)$
      for each vertex $w \in W$ **do**
        Update $E \leftarrow E \cup \{(p, w)\}$
        Update $V \leftarrow V \cup \{p\}$
return $G = (V, E)$

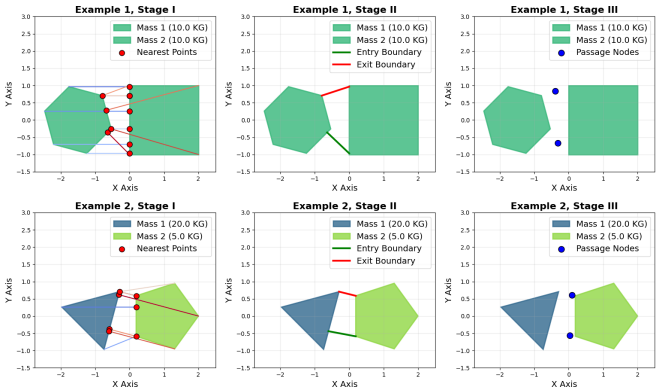


Fig. 3: Visualization of passage node construction between two sets of obstacles: (1) Hexagon and square (top row), and (2) Triangle and pentagon (bottom row). Each stage displays the progression from initial shape plotting to the identification of nearest points, boundary construction, and final passage node generation.

the boundary using linear interpolation, shifting it closer to the lighter obstacle to minimize contact with the heavier one. The interpolation factor γ depends on the masses m_i and m_j of the obstacles. A passage node $v_z \in V_{\text{passage}}$, where $z \in \{1, \ldots, n_{\text{passage}}\}$ and n_{passage} is the total number of passage nodes, is defined as:

$$v_z = v_i + \gamma(v_j - v_i), \qquad \text{where } \gamma = m_j \left(m_i + m_j\right)^{-1}. \quad (1)$$

In contrast to free space nodes, passage nodes have an additional cost based on the distances from the passage point to nearby obstacles, adjusted by their respective masses. Specifically, for a passage node $v_z \in V_{\text{passage}}$, the cost C_z is computed as:

$$C_z = \max\left(0, 1 - \frac{d_i}{r}\right) \cdot m_i + \max\left(0, 1 - \frac{d_j}{r}\right) \cdot m_j \quad (2)$$

The SVG constructs a weighted, undirected graph where edge costs represent travel distances between waypoints, and node costs reflect the effort required to manipulate obstacles (Fig. 2). A* [17] is used to compute the optimal path, guided by edge costs for robot movement and manipulation-effort cost of the passage nodes, with the Euclidean distance as a heuristic. This ensures that both the distance traveled and the physical manipulation required are accounted for in determining the lowest-cost path. Using the set of waypoints, we apply linear interpolation, represented as a first-order polynomial, to ensure uniform spacing between waypoints [26]. This spacing aids in normalizing the distance cost term for our local control strategy MPPI.

B. MPPI with Contact-Force Minimization

The method introduced in Sec. III-A generates a set of waypoints from the robot's starting position to the goal. In this section, we outline the constraints and objective function of MPPI, which ensure waypoint following while minimizing push actions by reducing contact forces on the robot. By integrating MPPI with the IsaacGym physics engine, as described in Sec. II-C, we extract the robot's contact force tensor and incorporate it into the objective function.

The objective function is subject to both equality and inequality constraints. The robot's state transition, $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{v}_t)$, for all $t \in [0, \dots, T-1]$, is computed using IsaacGym [19]. The initial state is constrained by the robot's current pose, $\mathbf{x}_0 = \mathbf{x}_{\text{init}}$ and inequality constraints ensure that the robot's state $\mathbf{x}_t \in \mathcal{X}$ and control inputs $\mathbf{v}_t \in \mathcal{V}$ remain within the feasible space for all $t \in [1, \dots, T]$.

MPPI defines state trajectories Q_k across $k \in K$ rollouts, each evaluated by a corresponding cost C_k. Unlike the implementation in [19], we omit the discount factor λ and concentrate most terms within the terminal cost C_{term} at the final prediction step rather than distributed across intermediate steps with stage cost C_{stage}. This ensures that MPPI emphasizes long-term planning and allows for deviations from the path at intermediate stages, to avoid unnecessary push actions. The total cost function C_{NAMO} for each rollout k is defined as:

$$C_{\text{NAMO}} = \begin{cases} C_{\text{term}} & \text{, if } t = T \\ C_{\text{stage}} & \text{, otherwise.} \end{cases} \tag{3}$$

Next, we will discuss the individual cost terms that make up the total cost function. Each cost term, such as C_{dist} for distance or C_{rot} for rotation, is a weighted version of its corresponding component. For example, the distance cost is expressed as $C_{\text{dist}} = w_{\text{dist}} \cdot c_{\text{dist}}$, where w_{dist} is the associated weight. These weights serve as hyperparameters that balance the different components of the robot's behavior. The stage and terminal costs are structured as follows:

$$C_{\text{term}} = C_{\text{ctrl}} + C_{\text{dist}} + C_{\text{prog}} + C_{\text{rot}} + C_{\text{force}}, \tag{4a}$$

$$C_{\text{stage}} = C_{\text{ctrl}}. \tag{4b}$$

The control cost c_{ctrl} minimizes deviations between the robot's predicted velocity $\dot{\mathbf{x}}_t$, obtained using IsaacGym, and the desired control input $\mathbf{v}_t$. This cost penalizes jerky movements and encourages smooth trajectories. The weight for the control cost is represented as a diagonal matrix W_{ctrl}, which assigns different penalties to each velocity component. The control cost is expressed as:

$$c_{\text{ctrl}} = \left(\frac{|\dot{\mathbf{x}}_t - \mathbf{v}_t|}{\mathbf{u}_{\max}} \right)^T W_{\text{ctrl}} \left(\frac{|\dot{\mathbf{x}}_t - \mathbf{v}_t|}{\mathbf{u}_{\max}} \right). \tag{5}$$

The distance cost c_{dist} encourages the robot to move toward the next waypoint. First, the set $\mathbf{D}_t = \{d_{1,t}, \dots, d_{n_p,t}\}$ collects the Euclidean distances between the robot's predicted state $\mathbf{x}_t$ and all waypoints $\mathcal{P}$. The closest waypoint $\mathbf{p}_i$ is identified by finding the index i that minimizes $\mathbf{D}_t$. The robot then targets the next waypoint $\mathbf{p}_{i+1}$, where the distance $d_{i+1,t}$ is normalized by α:

$$c_{\text{dist}} = \frac{d_{i+1,t}}{\alpha}, \quad i = \arg\min(\mathbf{D}_t). \tag{6}$$

The progress cost c_{prog} evaluates the advancement of each rollout k along the path $\mathcal{P}$. Rollouts that lag behind are penalized. The index $i_{T,k}$ of the closest waypoint at the terminal step T is computed for each rollout k and collected over all rollouts in the set $I_T = \{i_{T,1}, \dots, i_{T,K}\}$. Rollouts with lower waypoint indices are penalized, while those further along the path are rewarded. The progress cost is defined as:

$$c_{\text{prog},k} = 1 - \frac{i_{T,k} - \min(I_T)}{\max(I_T) - \min(I_T)} \tag{7}$$

The rotation cost c_{rot} aligns the robot's predicted orientation θ_t with the direction of the next waypoint θ_{target}. This alignment ensures smooth navigation, particularly in cluttered environments where a precise orientation is necessary for safety. The target angle θ_{target} is computed using the arctan function, which calculates the angle between the robot's position and the upcoming waypoint $\mathbf{p}_{i+1} = [p_x, p_y]^\top$. The rotation cost is normalized by π:

$$c_{\text{rot}} = \frac{|\theta_{\text{target}} - \theta_t|}{\pi}, \quad \theta_{\text{target}} = \text{arctan2}(p_y - \mathrm{x}_t^y, p_x - \mathrm{x}_t^x). \tag{8}$$

The force cost c_{force} penalizes excessive contact forces on the robot's joints during interactions with obstacles. IsaacGym computes the forces applied to each joint at every prediction step. Rollouts with fewer push actions accumulate less force, resulting in lower penalties over the prediction horizon T. The cumulative force f_{robot} is summed across all joints $j \in N$ and prediction steps T, and normalized:

$$c_{\text{force}} = \frac{f_{\text{robot}}}{\max(f_{\text{robot}}) + \epsilon}, \quad f_{\text{robot}} = \sum_{t=1}^{T} \sum_{j=1}^{N} f_{j,t}. \tag{9}$$

C. Movability Evaluation

The robot evaluates the movability of objects based on their perceived mass distribution in the environment. If an object initially classified as movable resists manipulation, its mass distribution is updated to the maximum threshold, reclassifying it as non-movable. Currently, a binary re-evaluation of movability is used to avoid excessive replanning, though it could be extended to a continuous re-evaluation for a robotic system by estimating the obstacle mass from physical data [23, 24]. Updating movability estimates and replanning are therefore initiated when specific conditions suggest that an object is non-movable. A timer mechanism monitors these conditions and starts a timer t_{monitor} when any are triggered. If the timer exceeds τ_{replan}, the robot captures a new snapshot of the environment, updates the movability data, and regenerates the graph and path to the goal.

The first condition for replanning is triggered when the robot's joint velocities drop close to zero; replanning occurs if the joint velocities fall below $\epsilon = 0.1$ m/s, indicating significant resistance. The second condition involves a deviation between actual and desired joint velocities, with a deviation greater than 75% ($\lambda = 0.75$) of the desired velocity signaling ineffective movement. Lastly, if the joint velocities are high but the robot's position remains stable, indicating slipping, this condition is met when the robot is stationary but joint velocities exceed $\mu = 0.1$ m/s. If none of these conditions are met, the timer resets, pausing the evaluation process until one of the conditions is activated again.

IV. Experiments and Results

This section outlines the experimental setup and results, validating the proposed SVG-MPPI algorithm in both simulated and real-world environments. The algorithm's performance is evaluated based on its ability to navigate toward obstructed goal positions, and we compare it against other established methods.

A. Experimental Framework

The experiments are conducted using a Dingo-O robot, a holonomic, velocity-controlled platform developed by Clearpath Robotics. In real-world trials, precise localization of the robot and obstacles is achieved through a Vicon motion capture system, with poses updated in Isaac-Gym at 25 Hz to ensure accurate MPPI rollouts. A maximum pushable mass of 30 kg is enforced, along with a 0.3-meter safety margin based on the robot's width (517 mm) to prevent collisions. The waypoint interpolation interval is set to 0.5 meters, striking a balance between computational efficiency and path smoothness. Replanning is triggered if the conditions outlined in Section III-C persist for more than 30 seconds. MPPI control operates with a time step of 0.08 seconds and a prediction horizon of 25 steps, covering 2 seconds of movement. Further experimental details are available in our GitHub repository.

We evaluate the algorithm in a simulated 4x8 meter cluttered room, where 20% of the space is occupied by movable obstacles and 5% by stationary obstacles where obstacles are randomized with respect to their positions, dimensions and masses (see Fig. 4). This randomly structured environment challenges the robot to navigate tight spaces, where pushing is often required if avoidance is unfeasible. The randomness ensures robust testing by simulating real-world conditions, making it an effective benchmark for adaptability, efficiency, and real-time replanning capabilities.

B. Benchmark Algorithms

To the best of our knowledge, no existing algorithm directly mirrors the SVG-MPPI approach. Therefore, we developed three comparative methods to benchmark our system, focusing on how obstacle movability is interpreted in both global path planning and local control strategies.

The first method, NVG (Normal Visibility Graph), treats all obstacles as stationary. The second, BVG (Binary Visibility Graph), uses a binary classification, distinguishing between movable and immovable obstacles. The third method, B-RRT (Binary Rapidly-exploring Random Tree), also applies a binary classification of movability but uses an RRT-based path planning algorithm. In this method, sampling within the safety margin of obstacles is permitted if their mass is below 30 kg, though no additional node costs are applied. The RRT approach follows the traditional structure as outlined in [27]. Fig. 4a-d illustrates how each planner interprets movability and demonstrates a sample scenario where a path is calculated from the starting position to the goal.

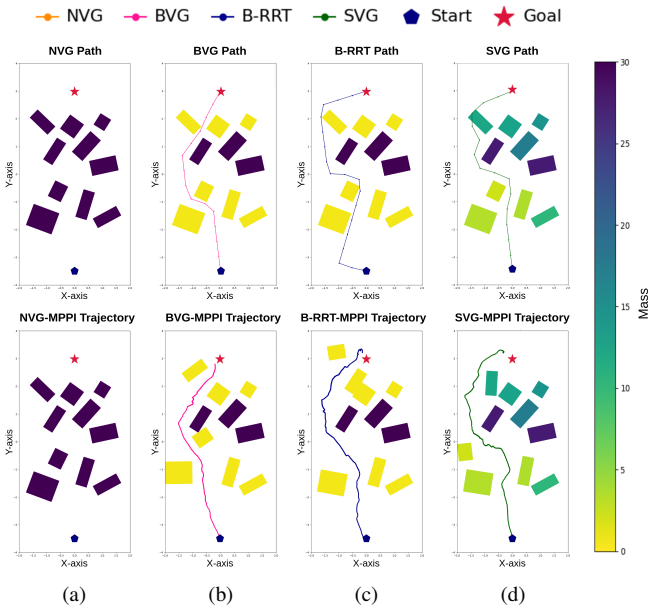


Fig. 4: Comparison of the different path planning strategies: (a) NVG, (b) BVG, (c) B-RRT, and (d) SVG. Each figure shows the planned path and the robot's trajectory using MPPI as the local motion planner. For NVG, no path could be found as all obstacles are considered non-movable.

In addition to comparing global path planning methods, we assess the impact of different movability interpretations on local control strategies by pairing each path planning method with MPPI. NVG-MPPI does not account for movability and treats all obstacles as stationary, while BVG-MPPI and B-RRT-MPPI apply a binary movability distinction in both the path and trajectory planner. In contrast, SVG-MPPI employs a continuous approach to movability. This comparison allows us to evaluate how different interpretations of movability affect both path planning and control performance.

C. Performance Metrics

Performance metrics were collected over 50 simulation runs, with obstacle masses randomly assigned between 4 and 36 kg. The path planners, NVG, BVG, B-RRT and SVG, are compared on their *Path Planning Success* indicating the percentage of paths found from start to goal and *Planner time* expressing the average computation time to generate the path. All planners are combined with MPPI as local motion planning method and analyzed on their *Execution Success* providing the percentage of scenarios where the simulated robot reaches the goal and its corresponding average *Execution time*. In addition, we include the cumulative force averaged over the 50 randomly generated scenarios to analyze the contact forces between the robot and obstacles over all successful trials.

D. Experimental analysis

In Table I, the performance metrics for SVG-MPPI are compared against benchmark planners as introduced in Sec. IV-B across 50 randomized scenarios. The proposed path planner SVG consistently outperforms NVG and B-RRT in path planning success rates with 98.2% for SVG and 16.4% and 92.7% for NVG and B-RRT respectively.

TABLE I: Experimental results of the compared methods for 50 scenarios with objects of randomized size, placement, and mass.

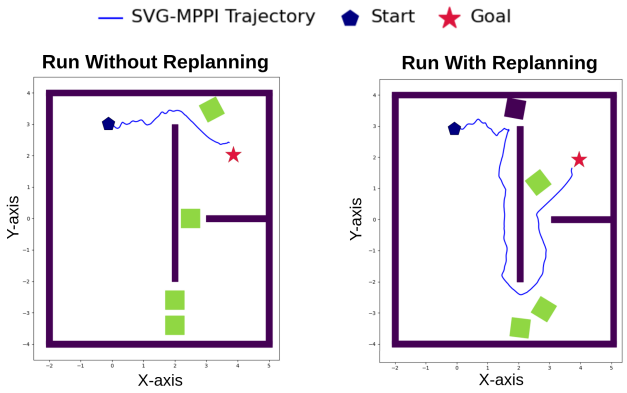
Path Planner	Path Planning Success (%)	Planner Time (s) (Mean ± SE)	Control Strategy	Execution to Goal Success (%)	Execution Time (s) (Mean ± SE)	Cumulative Force (kN) (Mean ± SE)
NVG	16.4	0.169 ± 0.014	NVG-MPPI	12.7	109 ± 18	39 ± 23
BVG	98.2	0.195 ± 0.006	BVG-MPPI	49.1	129 ± 13	165 ± 418
B-RRT	92.7	1.120 ± 0.483	RRT-MPPI	50.9	130 ± 9	175 ± 31
SVG	**98.2**	**0.196 ± 0.006**	**SVG-MPPI**	**54.5**	**120 ± 11**	**97 ± 17**

By integrating the continuous movability within the path planner SVG and the local planner MPPI, we obtain an improved execution success rate, 54.5%, compared to a binary classification method, 49.1% and 50.9% for BVG and B-RRT respectively, with a mildly improved execution time. Most importantly, SVG-MPPI also demonstrates significantly lower cumulative force compared to B-RRT-MPPI and BVG-MPPI, emphasizing its efficiency in minimizing push actions during navigation. Its strategic node placement, effort-based path selection, and considered continuous movability in both the local and global planner, lead to smoother interactions with the environment, making it particularly effective in scenarios requiring multiple obstacle manipulations. Note that NVG-MPPI can only solve scenarios where a collision-free path can be constructed. The minimal cumulative force in Table I results from unintended pushing due to suboptimal turns and path-following.

An illustrative example can be observed in Fig.4 expressing the paths and trajectories of all compared methods. By considering continuous movability, SVG-MMPI selects a path near the lighter obstacles that lead to a small displacement of the obstacles, e.g. a low push effort over the trajectory. B-RRT-MPPI, on the other hand, pushes one obstacle across the room, and both BVG-MPPI and B-RRT-MPPI push one obstacle against another obstacle resulting in additional push effort. NVG-MPPI is unable to generate a path in this environment as all obstacles are considered non-movable. SVG-MPPI applied replanning three times without success, but in two cases, it enabled the robot to reach the goal. Other methods did not include replanning. Moreover, SVG-MPPI's planner time remains highly efficient, 0.196 ± 0.006, supporting real-time replanning in dynamic environments. This combination of efficiency and adaptability highlights SVG-MPPI's robustness, particularly in environments with high obstacle density and frequent replanning needs.

1) Online Replanning: Fig. 5 demonstrates two scenarios where in scenario (a) the obstacle mass is as expected and in scenario (b) the obstacle is observed to be non-movable when encountered. In scenario (a), the robot moves a single obstacle to create a pathway along the upper side of the room. In scenario (b), the robot is unable to move the obstacle which triggers the replanning process as described in Sec. III-C. This allowed the robot to adapt by rerouting through the lower side of the room, demonstrating the ability of SVG-MPPI to update movability online and replan accordingly.

2) Real-World Experiments: In the real-world test, the Dingo-O robot navigates a hallway with movable and immovable obstacles to confirm the practical effectiveness of the SVG-MPPI algorithm. The robot successfully created a passage by relocating obstacles, reaching the goal despite the physical constraints of the environment, as shown in Fig. 1 and the accompanying video. During the real-world tests, the robot encounters three movable obstacles where the lightest obstacle, i.e. obstacle C, is moved forward and the robot navigates in the created free space between obstacles B and C. During the real-world experiments, obstacle B is slightly pushed due to model mismatch and additional disturbances in the real-world tests compared to the simulated scenario.



(a) (b)

Fig. 5: Comparison between (a) the trajectory without replanning, where the robot moves a single obstacle to create a path, and (b) the trajectory with replanning, triggered when the estimated movability is updated and the robot reroutes through another path.

V. CONCLUSION

Integrating obstacle movability into robot navigation enables access to previously unreachable areas through obstacle manipulation. We introduce SVG-MPPI, a framework for navigating environments with movable obstacles that minimizes push actions and avoids explicit obstacle placement. SVG informs MPPI by strategically placing nodes around movable obstacles using a continuous movability cost. Instead of relying on explicit contact and dynamics models, SVG-MPPI leverages the Isaac-Gym physics engine for real-time state transitions and contact force data to reduce push actions. Our proposed method outperforms other algorithms, including NVG-MPPI with no movability information, BVG-MPPI and B-RRT-MPPI with a binary notion of movability, in terms of success rate and contact force minimization. Future work could incorporate friction into the movability factor and introduce gain scheduling for more adaptive behavior in extreme movability cases.

REFERENCES

[1] Xuesu Xiao et al. "Motion planning and control for mobile robot navigation using machine learning: a survey". In: *Autonomous Robots* 46 (2020), pp. 569–597.

[2] John Reif and Micha Sharir. "Motion planning in the presence of moving obstacles". In: *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. 1985, pp. 144–154.

[3] M Stilman and J Kuffner. "Navigation among movable obstacles: real-time reasoning in complex environments". In: *4th IEEE/RAS International Conference on Humanoid Robots, 2004.* Vol. 1. 2004, pp. 322–341.

[4] Erik D Demaine, Martin L Demaine, and Joseph O'Rourke. "PushPush and Push-1 are NP-hard in 2D". In: *ArXiv* cs.CG/0007021 (2000).

[5] Kirsty Ellis et al. "Navigation Among Movable Obstacles with Object Localization using Photorealistic Simulation". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 1711–1716.

[6] Benoit Renault et al. "Modeling a Social Placement Cost to Extend Navigation Among Movable Obstacles (NAMO) Algorithms". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 11345–11351.

[7] Hai-Ning Wu, Martin Levihn, and Mike Stilman. "Navigation Among Movable Obstacles in unknown environments". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 1433–1438.

[8] Dennis Nieuwenhuisen, A F van der Stappen, and Mark H Overmars. "An Effective Framework for Path Planning Amidst Movable Obstacles". In: *Workshop on the Algorithmic Foundations of Robotics*. 2006.

[9] Elias Mueggler et al. "Aerial-guided navigation of a ground robot among movable obstacles". In: *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)* (2014), pp. 1–8.

[10] Nicola Castaman, Elisa Tosello, and Enrico Pagello. "A Sampling-Based Tree Planner for Navigation Among Movable Obstacles". In: *Proceedings of ISR 2016: 47st International Symposium on Robotics*. 2016, pp. 1–8.

[11] Shokraneh K Moghaddam and E Masehian. "Planning Robot Navigation among Movable Obstacles (NAMO) through a Recursive Approach". In: *Journal of Intelligent & Robotic Systems* 83 (2016), pp. 603–634.

[12] Zehui Meng et al. "Active Path Clearing Navigation through Environment Reconfiguration in Presence of Movable Obstacles". In: *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. 2018, pp. 156–163.

[13] Jose Muguira-Iturralde et al. "Visibility-Aware Navigation Among Movable Obstacles". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 10083–10089.

[14] Ewerton R Vieira et al. "Persistent homology for effective non-prehensile manipulation". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 1918–1924.

[15] Mark De Berg. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000.

[16] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[17] Peter Hart, Nils Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[18] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. "Model Predictive Path Integral Control: From Theory to Parallel Computation". In: *Journal of Guidance, Control, and Dynamics* 40.2 (2017), pp. 344–357.

[19] Corrado Pezzato et al. "Sampling-Based MPC Using a GPU-parallelizable Physics Simulator as Dynamic Model: an Open Source Implementation with Isaac-Gym". In: 2023.

[20] Viktor Makoviychuk et al. "Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning". In: *CoRR* abs/2108.10470 (2021).

[21] Trevor Scott Standley et al. "image2mass: Estimating the Mass of an Object from Its Image". In: *Conference on Robot Learning*. 2017.

[22] Jiajun Wu et al. "Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning". In: *Neural Information Processing Systems*. 2015.

[23] Sergio Aguilera et al. "Mass Estimation of a Moving Object Through Minimal Manipulation Interaction". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 6461–6467.

[24] Anirvan Dutta, Etienne Burdet, and Mohsen Kaboli. "Push to Know! - Visuo-Tactile Based Active Object Parameter Inference with Dual Differentiable Filtering". In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2023, pp. 3137–3144.

[25] Jiacheng Yuan et al. "Active Mass Distribution Estimation from Tactile Feedback". In: *ArXiv* (2023).

[26] Agarwal Ravi P and Patricia J Y Wong. *Spline Interpolation*. Dordrecht: Springer Netherlands, 1993, pp. 281–362.

[27] Steven M LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* (1998).