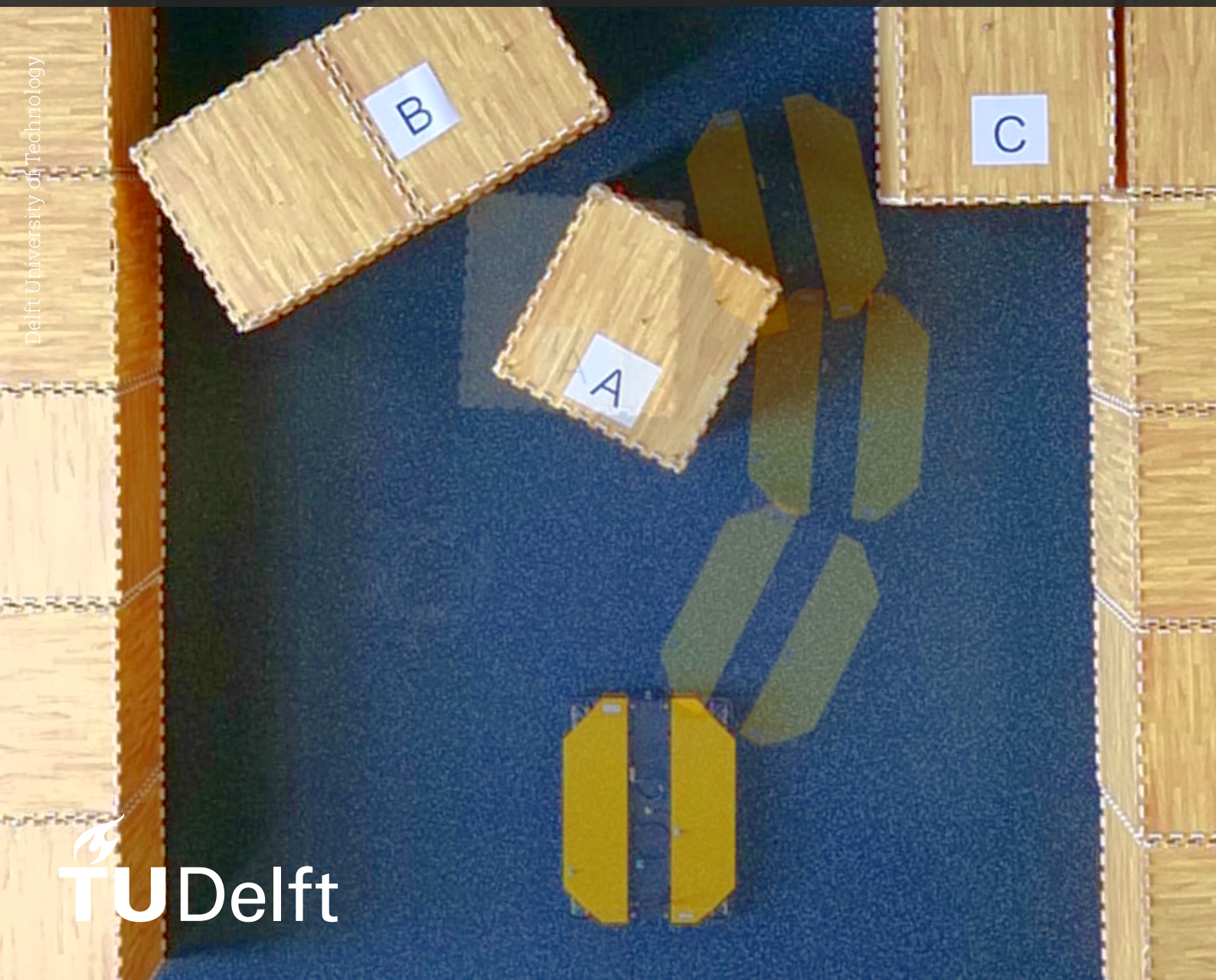
Improving Indoor Navigation Through Cluttered Rooms Using Movability Estimation

RO57035: MSc. Thesis

J. J. Weeda



Delft University of Technology

TU Delft

Improving Indoor Navigation Through Cluttered Rooms Using Movability Estimation

by

J. J. Weeda

To obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on:
28-08-2024

Version:	Final	
Student number:	5641551	
Thesis committee:	Dr. Javier Alonso-Mora	TU Delft, Primary supervisor
	Ir. Saray Bakker	TU Delft, Daily supervisor
	Dr. Clarence Chen	TU Delft, Daily supervisor
	Prof.dr.ir. Martijn Wisse	External committee member

An electronic version of the thesis is available at https://repository.tudelft.nl/

TU Delft

Abstract

Traditional path-planning methods for mobile robots typically focus on avoiding obstacles but often fall short when obstacles block the path to the goal. This paper addresses the challenge of Navigation Among Movable Obstacles (NAMO), where a single robot can reposition obstacles to create previously inaccessible pathways. We introduce SVG-MPPI, a novel approach that integrates semantics to incorporate continuous movability into both path planning and local control strategies, allowing the robot to navigate cluttered environments by moving obstacles as needed to reach its goal.

SVG-MPPI refines the traditional Visibility Graph (VG) method by introducing the Semantic Visibility Graph (SVG). In this advanced approach, additional nodes are placed near movable obstacles, with the movability of these obstacles assessed based on their estimated mass—a key semantic property defining movability. This enables the modeling of potential passages through these barriers. The cost of push actions needed to navigate to these additional nodes is incorporated into the path-finding algorithm, eliminating the need for explicit obstacle placement and reducing the overhead of custom task planning. The local control strategy employs Model Predictive Path Integral (MPPI), which uses the IsaacGym physics engine to simulate robot and obstacle state transitions. MPPI minimizes trajectory contact forces as part of its objective, thereby reducing the push actions executed by the robot. The system supports optional replanning by continuously evaluating obstacle movability and adjusting the path if actual conditions deviate from initial estimates.

Our solution was evaluated through both qualitative and quantitative experiments. Qualitative experiments demonstrated the algorithm's success in a simulated environment, where it effectively handled path execution and replanning scenarios. In real-world testing, an omnidirectional robot successfully pushed an obstacle and established a path to a goal. Quantitative analyses compared SVG-MPPI with its unaltered planning method (VG), which has no notion of movability, and Rapidly-exploring Random Tree (RRT) adapted to include binary movability. Each of these path planning methods was equipped with MPPI, which had a similar notion of movability as the planner. Results showed that SVG-MPPI consistently outperformed both VG and RRT in terms of path planning success rate and execution success rate. Furthermore, SVG-MPPI exhibited lower cumulative contact forces over the trajectory, indicating that the integration of continuous movability allowed the algorithm to select paths of least resistance and navigate around obstacles where possible.

Overall, SVG-MPPI represents a significant advancement in NAMO planning by offering a cohesive solution that addresses the complexities of navigating random, cluttered environments with movable obstacles. By prioritizing direct progress toward the goal while repositioning obstacles, SVG-MPPI introduces a novel, integrated approach with promising results and substantial potential for future research and development.

Acknowledgements

A huge thanks to the AMR team at TU Delft for the opportunity to complete my thesis there. A special appreciation to my daily supervisors, Saray and Clarence—your guidance, support, and enthusiasm made this journey much smoother and more enjoyable. I couldn't have done it without your expertise and encouragement! To my wonderful girlfriend, Charlotte, your patience and belief helped me get through the busy times with a smile. And to my family and friends, especially my brother Jesse, who was always there to nudge me in the right direction—your support has been invaluable. I hope you find this work as rewarding to read as I found it to create.

J. J. Weeda
Delft, August 2024

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

The deployment of modern service robots is rapidly expanding across both human-centered and non-human-centered environments, impacting society as a whole. These robots, designed to assist with tasks traditionally carried out by humans, feature advanced perception and adaptive behavior capabilities, revolutionizing our interaction with technology [1].

In domestic environments, service robots handle everyday chores, allowing individuals to allocate their time more freely to focus on meaningful activities [2]. They are also employed in other human-centered environments such as retail stores, office buildings, and educational settings [3–5]. In these settings, robots assist customers in stores, manage reception areas, and serve as teaching assistants in classrooms. In healthcare and hospitals, service robots can help care for the elderly, children, and individuals with disabilities, thereby easing the workload of caregivers and addressing personnel shortages [6, 7].

Figure 1.1, shown on page 2, presents a multi-purpose service robot[1] designed for use in a typical cluttered domestic environment. This robot can be instructed to perform everyday tasks such as cleaning, organizing, and retrieving items.

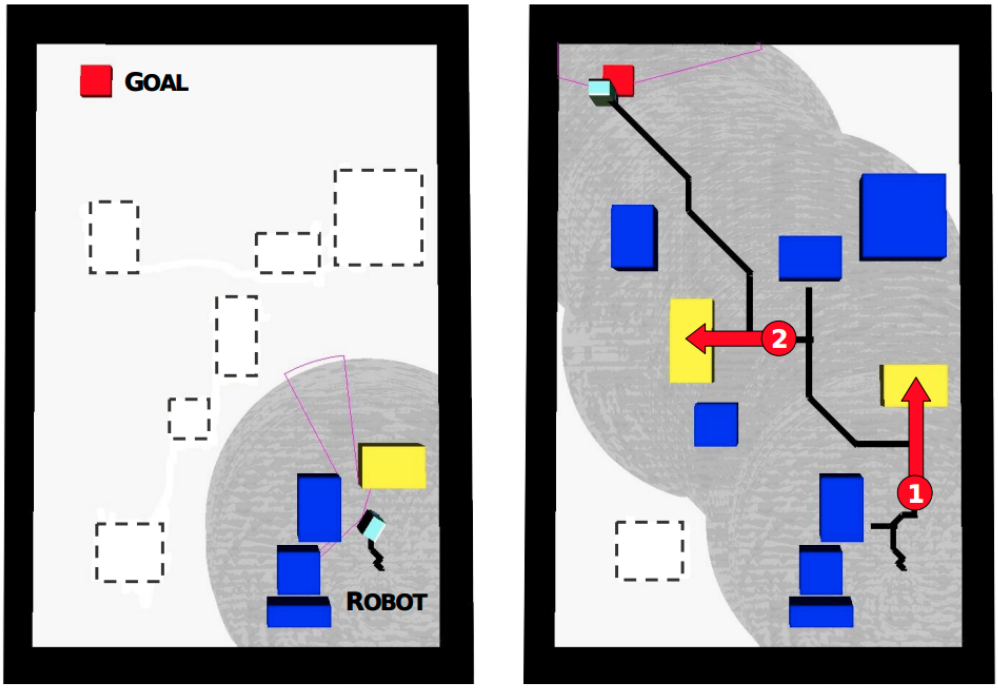
In non-human-centered environments, service robots operate autonomously, performing tasks without direct human intervention or interactions. In agriculture, robots manage greenhouses by monitoring plant conditions, managing harvests, and applying precision techniques for pest and weed control [8, 9]. In industrial settings, they inspect critical infrastructure such as pipelines and energy grids to detect damage and ensure safety [10]. Their roles also extend to surveillance, security, reconnaissance, and search-and-rescue operations [11, 12].

For these autonomous service robots, navigation typically involves moving towards specific waypoints while avoiding collisions [13]. But despite their versatility, these robots often face challenges in complex and cluttered environments, particularly in domestic settings [14, 15]. This is where Navigation Among Movable Obstacles (NAMO) becomes relevant. NAMO involves navigating environments where obstacles can be moved to clear a path, requiring interaction with the environment and thereby relocating obstacles to create previously blocked pathways [16–18]. Occupation-based representations work well for basic obstacle avoidance. However, they fall short when obstacles block the path to the goal as characteristics like size, shape, or mass of individual objects is ignored [19].

Semantic mapping offers a promising solution by providing a detailed representation of the environment, including information about objects, structures, and features [20]. This enriched representation improves the robot's ability to understand and interact with its surroundings. However, the potential of semantic mapping in addressing NAMO has not yet been explored. This study presents an algorithm that leverages semantic information to infer obstacle movability, aiming to improve navigation efficiency and effectively handle complex cluttered scenarios. This advancement could significantly enhance autonomous navigation and the deployment of service robots.

[1] https://www.unlimited-robotics.com/gary

Figure 1.1: The multi-functional service robot 'Gary', employed in a cluttered domestic environment.



(a) Initial State (b) Final Execution

Figure 1.2: Achievement of a simple NAMO task under partial observed information. Movable objects are yellow (light), while immovable ones are in blue (dark). In (a), dashed lines signify unknown objects. (b) illustrates the progression of the robot's internal map [18].

1.1. Related Work

Navigation Among Movable Obstacles (NAMO) involves a single robot manipulating objects within the environment to create previously inaccessible pathways. The study of NAMO dates back to research conducted by Reif and Sharir in 1985 [21]. This early work considered a two-dimensional polygonal workspace, modeling both the movable object and robot as convex polygons. This study demonstrated the computational complexity by proving that determining whether a robot can reach a specified goal position in the presence of movable obstacles is NP-hard. This complexity persists even in simplified scenarios with unit square obstacles, requiring extensive computational resources to explore all potential configurations and movement sequences to ensure the robot can navigate effectively to the goal [21, 22]. Navigation Among Movable Obstacles, as a category of research, was formally introduced by Stilman and Kuffner [17] in 2004 as a practical extension for humanoid and dexterous mobile robots, enabling them to tackle complex tasks where the goal pose is not directly reachable.

The NAMO problem, even in its simplest form, is classified as NP-Complete [22]. This complexity stems from its vast search space and the large number of potential configurations to check as a possible solution. Consequently, most solutions focus on approximations rather than exact solutions. Despite the importance of this problem, fully autonomous NAMO systems are rare in mobile robotics research [23], with even fewer examples of real-world applications.

Therefore, many studies focus on addressing a subset of the NAMO problem, known as LP_1. In LP_1 scenarios, two disconnected segments are blocked by a single obstacle. By manipulating this obstacle using basic repositioning techniques such as pushing, the robot can create a path towards the goal. This approach reduces the problem's complexity by focusing on the navigational challenge and repositioning of a single object. Research also addresses solutions to the LP_2 problem, in which up to two objects may be moved multiple times to connect free space components for the robot and achieve the final positions of the obstacles. This approach can be further generalized to LP_k, where k obstacles are involved. Other variations focus on more complex aspects such as partial environmental data or socially aware placement of the obstacles [17, 18, 24].

Figure 1.2, shown on page 2, illustrates a straightforward example of NAMO, highlighting a scenario where reaching the goal point necessitates altering the environment [18]. Movable objects are shown in yellow (light), while immovable ones are in blue (dark). Initially, one mandatory manipulation is necessary to create a path to the goal. Subsequently, a second manipulation is performed to create a faster route to the goal, eliminating detours and reducing the traveled distance. In this example, movability is inferred during execution by applying a push action executed by the robot. When a push action fails, the obstacle is considered to be static, and the task planner will generate an alternative route to the goal.

Solutions for Navigation Among Movable Obstacles (NAMO) that deal with a variety of obstacles, denoted as LP_k, usually depend on specialized task planners to direct the robot's actions. The task planner determines instances when the robot should move without displacing an obstacle (transit) and when it should relocate an obstacle (transfer). While these task planners are tailored to solve specific subsets of the problem, they often make several assumptions to streamline the algorithm and simplify the problem.

Table 1.1 provides a comprehensive overview of NAMO algorithms found in the literature, highlighting key criteria related to prior knowledge and implementation strategies. Three important observations can be drawn from the table. First, the movability of an object is frequently either given or detected using simple obstacle detection techniques such as QR codes or color codes. This movability is typically treated as a binary property, which fails to account for the varying characteristics of obstacles that influence the ease or difficulty of interaction. Such a binary simplification complicates minimizing energy, time, or effort. Algorithms aiming to reduce effort during the execution of NAMO typically quantify it by the number of obstacles moved and the length of the route taken.

The second observation centers around task planners; current solutions often rely on custom-designed planners, indicating a lack of generalizable or reproducible approaches. Finally, many algorithms are primarily tested in simulated environments, with limited real-world demonstrations. This gap highlights the need for more practical evaluations to validate the effectiveness of these algorithms.

Integrating semantic information into NAMO algorithms has been a relatively unexplored topic. Traditionally, navigation maps account only for free and occupied spaces, lacking detailed contextual information about the objects within the environment. A semantic map, however, provides a richer representation by incorporating both *what* objects are (semantic information) and *where* they are located (geometric information). This combination of semantic and geometric data offers valuable insights into the general characteristics of objects, which can be useful for the robot, especially in NAMO.

By understanding the nature of the obstacles, whether they are furniture, appliances, or other entities, robots can make more informed decisions about how to interact with their environment. For example, knowing whether an obstacle is a chair or a table can influence the robot's strategy for moving it or navigating around it. Semantic information can also help in predicting the potential movability of objects and in planning the most efficient path with minimal energy expenditure.

Table 1.1: Overview of differentiating NAMO algorithms.

Author & Reference	Env.	Mov.	Quant.	Cost	C-Space	Task Planner	Transit Planner	Transfer Planner	Demo
Stilman et al. [17]	Full	Given	No	Energy	Disc.	DFS	A*	BFS	No
Wu et al. [18]	None	Man.	No	-	Disc.	Custom	D*lite	DFS	No
Nieuwenhuisen et al. [25]	Full	Given	No	Distance	Cont.	Custom	RRT	RRT	No
Mueggler et al. [26]	Full	Rec.	No	Time	Disc.	Custom	A*	Dijk	Yes
Castaman et al. [27]	Full	Given	No	Time	Disc.	KPIECE	A*	A*	No
Moghaddam et al. [28]	Full	Given	No	Energy	Cont.	DFS	Dijk + VG	Dijk + VG	No
Meng et al. [29]	Part.	Rec.	No	Distance	Cont.	Custom	RRT	RRT	Yes
Ellis et al. [23]	Part.	Man.	No	Distance	Cont.	Custom	A* + VG	-	No

Legend: '-' = Not Found; '+' = Combination; Ref = Reference; Env = Environment; Quant = Quantified; C-space = Configuration-Space; Man = Manipulation, Part = Partial, Rec = Recognized, Disc = Discrete, Cont = Continuous, Demo = Real World Demonstration

1.2. Contribution

This work demonstrates the integration of semantic information into the NAMO-solving process, enhancing decision-making regarding object manipulation to achieve goal positions effectively. Inspired by the framework proposed by Ellis et al. [23], which uses IsaacSim as a physics engine to model state transitions of the environment, we introduce the Semantic Visibility Graph combined with Model Predictive Path Integral (SVG-MPPI). This method accommodates NAMO without explicit obstacle placement or the need for a task planner, distinguishing it from the solutions outlined in Table 1.1. The main contributions of this work include:

1. **Integration of Continuous Quantified Movability**: We introduce quantified movability of obstacles on a continuous scale instead of binary classification. Supported by semantic information, this quantified movability is included in the global path planner as a node cost in the weighted semantic visibility graph (SVG).
2. **Contact-Force Minimization using MPPI**: Our approach incorporates the concept of movability into the local planner by minimizing contact forces between the robot and the environment while following the trajectory from start to goal.
3. **Elimination of Explicit Obstacle Placement**: We remove the need for explicit obstacle placement used in existing NAMO methods, thereby eliminating the overhead of a task planner for defining the sequence of actions.
4. **Replanning on Movability Evaluation**: We evaluate the quantified movability of obstacles during interactions and replan if the evaluation does not match the initial estimation.

1.3. Overview

First, Chapter 2 will cover important preliminaries to provide the necessary background information on the topics and algorithms addressed in this thesis. Specifically, the widely used Visibility Graph will be explained, along with the MPPI control algorithm used to determine control actions. Following this, Chapter 3 will detail the modifications to components of the proposed SVG-MPPI algorithm, explaining and motivating each change. Next, in Chapter 4, the solution is tested in various simulated environments to showcase and quantify its behavior with a final demonstration in a real life scenario. The results of the test are then discussed in Chapter 5. Finally, Chapter 6 will conclude, accompanied by a critical discussion of the results and potential follow-up actions to further extend this work.

2

Preliminaries

This chapter provides essential background information for the thesis. It starts with an overview of the traditional Visibility Graph (VG) algorithm, highlighting both its strengths and limitations. The discussion then moves to the Model Predictive Path Integral (MPPI) control strategy used in IsaacGym[1], a physics simulation environment responsible for computing system dynamics and determining control actions for the robot.

2.1. Visibility Graphs (VG)

A Visibility Graph represents intervisible locations for an agent, forming a weighted graph where each node corresponds to a point location and each edge denotes a visible connection between them [30]. This graph provides a way of constructing a set of nodes and edges between a starting node and a goal node, facilitating path planning algorithms like Dijkstra [31], Bi-directional Dijkstra [31], or A* [32].

Visibility graphs inherently address the challenge of navigating around stationary obstacles by maintaining a sufficient distance from each obstacle within the environment. Nodes are placed at a safe distance from the obstacles, where the safety margin is considered a hyperparameter and often depends on the robot [33]. Edges connect nodes that have a direct line of sight or visibility to each other without passing through any obstructing obstacles. This direct visibility ensures that paths in the graph are obstacle-free, minimizing detours to find a path towards the goal.

Algorithm

In Algorithm 1, the construction of the visibility graph is summarized on which we will later build our semantic visibility graph in Section 3. Here, V represents the set of all vertices in the visibility graph, while E denotes the set of edges connecting these vertices. The algorithm iterates through each polygon S_i in the set of obstacles S, inflates it to account for a safety margin r, and extracts its vertices P_i. For each vertex v in P_i, the algorithm identifies visible vertices W using the *visibleVertices* method. Next, it adds edges connecting v to each visible vertex w in the set of edges E and then adds v to the set of vertices V. The algorithm concludes by returning the constructed visibility graph $G = (V, E)$.

The method *visibleVertices* takes as input a set S of polygonal obstacles and a point v in the plane (typically a vertex of one of the obstacles in S) and outputs all obstacle vertices that are visible from v. A vertex w is deemed visible from v if the segment (v, w) doesn't intersect any obstacle's interior. This requires searching with w along the obstacle edges intersected by the half-line starting at v and extending through w. If the edge does not intersect any of the obstacles, it is considered to be in free space and allowed to be added to the set of edges E. Applying the *visibleVertices* method to all vertices of all polygons implies full visibility for the algorithm, enabling the construction of the entire map.

[1] https://developer.nvidia.com/isaac-gym

Algorithm 1: Visibility Graph [30]

input A set S of disjoint polygonal obstacles.
output The visibility graph $G(S)$.
$V \leftarrow \emptyset; E \leftarrow \emptyset$
for $i = 0, \ldots, |S|$ **do**
    $P_i \leftarrow$ vertices of inflated polygon S_i with safety margin r
    for $j = 0, \ldots, |P_i|$ **do**
        $v \leftarrow P_i[j]$
        $W \leftarrow$ visibleVertices(v, S)
        for *each vertex* $w \in W$ **do**
            $E \leftarrow E \cup \{(v, w)\}$
        $V \leftarrow V \cup \{v\}$
return $G = (V, E)$

limitations

Visibility graphs possess inherent limitations. Suppose S comprises a collection of disjoint polygonal obstacles within a plane, amounting to n boundaries. Constructing a visibility graph for S means identifying pairs of vertices that share direct line-of-sight connections. Initially, this detection can be performed in $O(n)$ time per vertex. However, fully evaluating all potential vertex pairs escalates the complexity to $O(n^3)$, placing substantial computational demands [30]. Over time, several improvements have been proposed to mitigate the time complexity of this naive search algorithm. Examples include the radial scanning algorithm by Lee [34], the plane-sweep algorithm introduced by Ghosh and Mount [35], and the convex-point approach suggested by Priya and Sridharan [36]. More modern methodologies include the triangular approach Ganguli, Cortés, and Bullo [37] and the use of geometric spatial querying Masud et al. [38].

Another significant drawback arises when the environment contains disconnected regions due to obstacle barriers. In such cases, not all locations are visible to each other, leading to gaps in the visibility graph. For instance, if obstacles surround the goal position, the visibility graph algorithm fails to find a path because there are no feasible edges connecting nodes within the visible region to the goal. This limitation prevents the algorithm from providing a solution in environments where the goal is entirely enclosed by obstacles, effectively isolating it from the start position or any other reachable area.

2.2. Model Predictive Path Integral (MPPI)

Model Predictive Path Integral (MPPI) control is a sophisticated control strategy designed to solve stochastic optimal control problems in discrete-time dynamical systems. MPPI leverages a sampling-based approach to determine optimal control actions, making it particularly effective in complex or uncertain environments [39].

The core of MPPI involves generating multiple sequences of control inputs, each perturbed by noise to explore a wide range of possible actions. These sequences are then used to simulate potential trajectories of the system using a system model over a specified time horizon. A cost function evaluates each trajectory based on criteria such as distance to waypoint and obstacle collisions. The trajectories with lower costs are given higher importance, and the control inputs that generate these trajectories are weighted more heavily. This process iteratively refines the control strategy, guiding the system toward optimal performance while accounting for uncertainties in the environment and system dynamics.

In the works presented by Pezzato et al. [40], MPPI has been implemented using the physics engine provided by IsaacGym. MPPI provides the control framework, leveraging its sampling-based methodology to identify high-performing control actions, while IsaacGym simulates the robot's behavior and its environment. Leveraging IsaacGym's parallel environmental setup, which uses GPU power for concurrent computing, multiple action samples, also known as rollouts, can be simulated simultaneously, drastically reducing computation time for the robot.

Algorithm

The integration of MPPI with IsaacGym is detailed in Algorithm 2. The algorithm starts by initializing an input sequence U_{init} as a vector of zeroes, with a length corresponding to the time horizon T in steps. To explore the input space, K sequences of additive input noise E_k are sampled using the Halton Sequence, chosen for its ability to enhance exploration and produce smoother trajectories [41].

The environment state x is observed, and all rollouts are reset to this state. The sampled noise E_k is added to U_{init}, which represents the best control action from the previous sequence. Using these action sequences, rollout costs are computed. IsaacGym facilitates this by computing the next state x_{t+1} from x_t and control input v_t, avoiding the explicit definition of system dynamics $f(x_t, v_t)$.

In the *computeRolloutCost* function, sampled input sequences V_k are used to simulate state trajectories Q_k over a planning horizon T. The cost of each trajectory S_k is then calculated using a cost function C, as detailed in Equation (2.1). This cost function measures the performance of the system, aiming to be minimized. To emphasize the importance of earlier actions, the cost is discounted using a factor γ, making early actions more significant in the overall evaluation.

Once the costs are determined, the function calculates importance sampling weights w_k to rank the quality of each rollout, as shown in Equation (2.2). The weights are computed based on the cost values, with the normalization factor η ensuring that the weights sum to 1. The parameter β adjusts how sensitive the weights are to differences in cost S_k, influencing how the importance is distributed among the rollouts. The term ρ represents the minimum sampled cost, helping to keep the weights positive and properly scaled.

$$S_k = \sum_{t=0}^{T-1} \gamma^t C(\mathbf{x}_{t,k}, \mathbf{v}_{t,k}) \tag{2.1}$$

$$w_k = \frac{1}{\eta} \exp\left(-\frac{1}{\beta}(S_k - \rho)\right), \quad \sum_{k=1}^{K} w_k = 1 \tag{2.2}$$

The optimal control sequence U^* is approximated as a weighted average of the sampled inputs, with the weights reflecting their respective importance. Once U^* is computed, it updates the initial control sequence U_{init} for the next iteration. The first action u_0^* of the new sequence U^* is applied to the system, and this process repeats until the task is completed. Leveraging the GPU capabilities of IsaacGym, all rollouts can be computed in parallel, significantly reducing computation time and enhancing the efficiency of the robot's control.

Algorithm 2: MPPI Using IsaacGym [40]

Initialize:
$U_{\text{init}} = [0, \ldots, 0]$
$E_k \leftarrow$ sampleHaltonSplines()
while *taskNotDone* **do**
  $x \leftarrow$ observeEnvironment()
  resetSimulations(x)
  for $k = 1$ **to** K **do**
    $V_k \leftarrow U_{\text{init}} + E_k$; // in parallel
    $[Q_k, S_k] \leftarrow$ computeRolloutCost(V_k, γ) ; // in parallel
    $w_k \leftarrow$ importanceSampling ; // in parallel
  $\beta \leftarrow$ updateBeta(β, η)
  $U^* \leftarrow \sum_{k=1}^{K} w_k V_k$
  $U_{\text{init}} \leftarrow$ timeShift(U^*)
  applyInput(u_0^*)

2.3. Semantics To Object Movability

Affordances, a concept introduced by psychologist Gibson, describe the potential interactions between an agent and its environment, based on the object's properties and contextual factors [42]. These interactions are inherently subjective and vary with the capabilities of the agent. For instance, humans can intuitively recognize that a ball can be rolled, while the ease of this action depends on specific characteristics of the ball, such as its material, size, and density. Affordance is not exclusive to humans, in robotics affordances are extended to how robots interact with their environment [43, 44].

Semantic information is essential for interpreting affordances, as it involves understanding an object's properties and its roles in different contexts [45]. For instance, a flat surface can afford different uses depending on its context. In a home, a table surface is used for placing and organizing items, aided by surrounding furniture. Outside, the same flat surface might be used for walking or resting, influenced by its environment, like cars and trees. This shift in utility can be understood through contextual information, which considers the surrounding environment, or semantic information, which relates to the object's functional roles.

In practice, affordances are often simplified into binary categories [46], which can obscure important details such as the costs associated with actions, including energy and time. This simplification can be problematic as it fails to account for the full range of interactions between objects and agents. For instance, categorizing a ball simply as ”rollable” overlooks the differences between various types of balls, such as a football versus a bowling ball. This limitation highlights the need for a more nuanced approach that incorporates semantic understanding to capture the complexities of object interactions.

Estimating Movability

In the context of NAMO, affordances often translate to the concept of movability, which can be further simplified to pushability for robots without manipulators [47]. For example, when a robot encounters an obstacle like a box, labeling it as ”movable” does not account for the effort required to move it, which depends on factors such as the box's size, mass, inertia, and ground friction [44]. The agent's capabilities are leading in determining the affordance; an adult might easily move a box, while a child might struggle. Therefore, the semantic understanding of movability involves recognizing how various factors influence the effort required for manipulation.

Estimating movability is an area that has not been extensively explored in the literature. Both learning-based and non-learning methods usually focus on determining the probability of movability but often neglect the associated costs or handling difficulties. Non-learning methods, such as those using geometric primitives extracted from point cloud data [48] or pose graphs to monitor object behavior [49], offer valuable insights but may not fully capture the nuances of cost and semantic context.

Learning-based methods, while versatile, often emphasize human-centric affordances, complicating their application to robots. For instance, models like AffordanceNet [50] are designed to detect affordances such as graspability but are typically trained on datasets that focus on human interactions. This human-centric focus can make it challenging to adapt these models for robots. Recent advancements, such as visual interaction models [51] and object-based representations [52], show promise but may require extensive data and might not fully address the nuances of movability for different robots.

Evaluating Movability

Evaluating movability is well-documented, with clear distinctions between analytical and learning-based models. Analytical methods, including those based on classical mechanics for planar pushing [53] and friction estimation [54], rely on strong assumptions such as quasi-static conditions and simplified friction models. While foundational, these methods might not always capture the complexities of real-world interactions.

On the other hand, learning-based approaches offer greater flexibility. Recent advancements in reinforcement learning and neural physics engines [55, 56] provide dynamic methods for understanding object manipulation. Techniques incorporating tactile feedback [57] and interactive learning models, such as the ”push to know” framework [58], represent significant progress. These methods enhance the robot's ability to predict and adapt to the costs of object manipulation, offering a more nuanced understanding of movability.

3

Methodology

This section introduces the Semantic Visibility Graph (SVG) combined with the Model Predictive Path Integral (MPPI), abbreviated as SVG-MPPI. This solution is specifically designed for navigating indoor environments without requiring explicit object placement to establish pathways toward the goal. The chapter provides a detailed overview of the solution's procedural steps and integral components.

Figure 3.1 clarifies the solution by outlining three primary components: SVG, Shortest Path, and MPPI. Within the SVG framework, the environment is represented as a 2D map with discrete objects characterized by semantic attributes. A weighted graph is constructed based on the spatial properties of obstacles, including their size and movability. This graph serves as the basis for applying a shortest-path algorithm to compute waypoints from the initial position to the goal.

Movability is extended to a continuous value, allowing certain objects to be more favorable to move compared to others. Mass is identified as the primary semantic property for estimating movability. In practice, this property can be estimated based on object categories or determined using advanced learning methods such as Image2Mass [59] and Galileo [60]. Additionally, mass can be assessed through object interactions using learning-based techniques, including tactile feedback [57], the "push to know" model [58], or by observing object dynamics during manipulation [61]. It is important to note that deriving mass from environmental data is beyond the scope of this work, and therefore mass distributions are assumed to be known a priori.

The mass of an object is represented as a Gaussian distribution with a mean and standard deviation for each object category. If the mass is below the threshold that the robot can handle for pushing, the object is deemed movable, with lighter objects being more favorable. Friction is assumed to be constant across all objects. However, the strategy can be improved if additional semantic information is available.

The MPPI strategy directs velocity commands for the robot. Using a customized objective function and adhering to specified constraints, the control actions guide the robot through waypoints while minimizing manipulation efforts. When manipulation fails, the algorithm adjusts its assessment of obstacle movability and recalculates the path generation process accordingly. The execution concludes either when no path to the goal can be found or when the goal is reached.

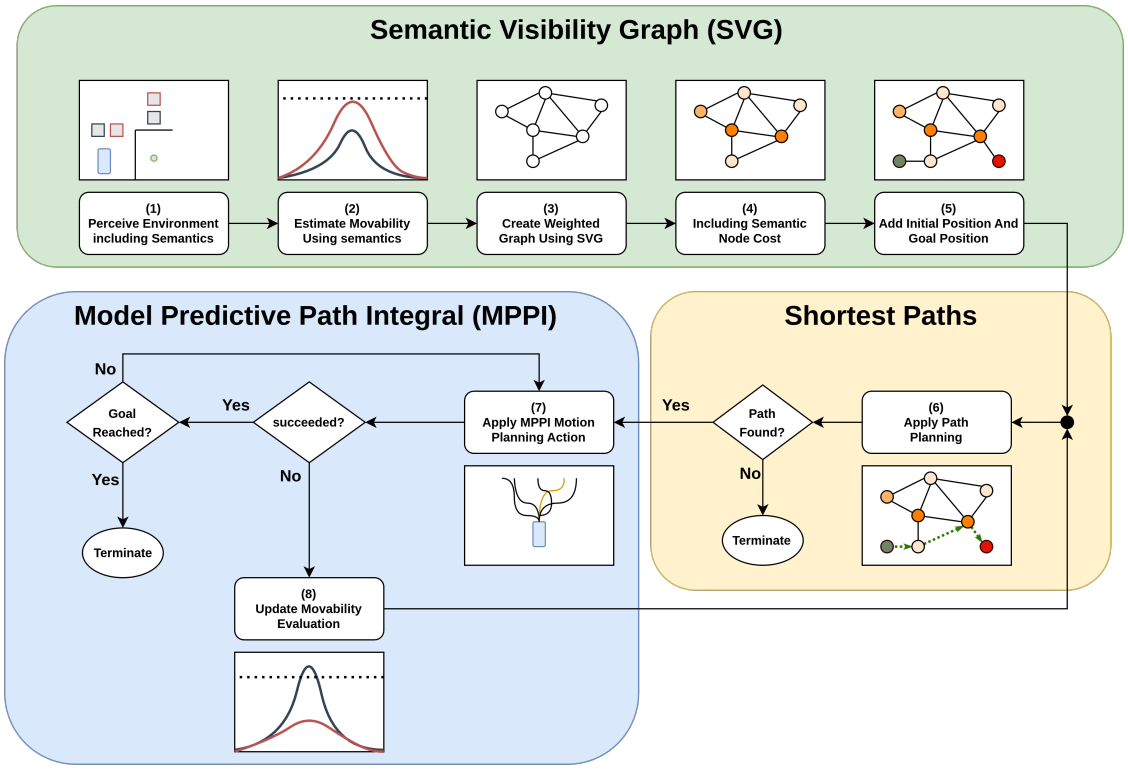


Figure 3.1: Overview of the SVG-MPPI solution to navigate the environment.

3.1. Semantic Visibility Graph (SVG)

This section introduces an extension to traditional Visibility Graphs (VG) that incorporates continuous movability. Traditional VGs are limited when obstacles obstruct parts of the map, hindering progress toward the goal. To overcome this limitation, the Semantic Visibility Graph (SVG) adds new nodes based on the (semantic) movability of obstacles. This alteration involves strategically placing extra nodes near movable obstacles and including additional node costs for push actions needed to access these nodes. These new *passage nodes* connect previously disconnected sets of nodes and vertices, thereby creating new pathways. Details of the modified algorithm can be found in Algorithm 3.

The algorithm's initial phase, lines 1-11 in Algorithm 3, follows the original approach of VGs outlined in Chapter 2. Obstacles are inflated to include a safety margin, and nodes are positioned at the vertices of these free-space polygons. Each node connects to visible vertices, ensuring that edges avoid intersecting other obstacles, resulting in vertices and edges being at a safe distance from the obstacles.

The subsequent phase, lines 12-19 in Algorithm 3, introduces passage nodes strategically placed near movable obstacles. First, unique combinations of obstacles on the map are identified. Obstacles that do not overlap when inflated with the safety margin are considered sufficiently apart for the robot to pass between. Pairs where the inflated obstacles overlap, meaning the distance between the two obstacles is below twice the safety margin, are too close for the robot to pass through.

Pairs considered too close to pass between, and with at least one movable obstacle (mass below the allowed threshold), can be processed for a passage. These newly created passage nodes are integrated into the weighted visibility graph similarly to existing nodes, establishing connections through visible vertices. As a result, previously disconnected regions are now linked, allowing shortest-path planning to reach previously inaccessible areas. However, accessing these passage nodes requires the robot to manipulate an obstacle, adding cost to visiting these nodes.

The specifics of node placement and visitation costs for passage nodes will be explained in subsequent sections. Following this, a shortest path algorithm is applied to the weighted graph to determine a path to the goal, considering both obstacle manipulation and movement between waypoints.

Algorithm 3: Semantic Visibility Graph

```
1  Input: A set S of disjoint polygonal obstacles.
2  Output: The visibility graph G(S).
3  V ← ∅; E ← ∅
4  for i = 0, ..., |S| do
5      P_i ← vertices of inflated polygon S_i with safety margin r
6      for j = 0, ..., |P_i| do
7          v ← P_i[j]
8          W ← visibleVertices(v, S)
9          for each vertex w ∈ W do
10             E ← E ∪ {(v, w)}
11         V ← V ∪ {v}
12 for each pair (P_i, P_j) where i ≠ j do
13     if distance between P_i and P_j < 2r then
14         if mass(P_i) ≤ max_mass or mass(P_j) ≤ max_mass then
15             p ← passageNodes(P_i, P_j) ;          // Generate Passages
16             W ← visibleVertices(p, S) ;           // Connect Passages
17             for each vertex w ∈ W do
18                 E ← E ∪ {(p, w)}
19             V ← V ∪ {p}
20 return G = (V, E)
```

3.1.1. Passage Nodes

This section describes a method for strategically placing passage nodes to connect areas obstructed by movable obstacles. The primary objective of these nodes is to link disconnected regions of the visibility graph, thereby facilitating paths through blocked waypoints. This method assumes that the robot has complete visibility of the entire map or room and can identify objects using basic semantic information, such as their class and mass.

Initially, inspiration is drawn from the Separating Hyperplane Theorem [62], which states that if two convex sets are disjoint, there exists a hyperplane that separates them. This theorem helps to understand that a line or space can exist between the obstacles. However, even with such a space or line, the exact node placement is not yet defined. Therefore, a different method for calculating entry and exit points between two convex sets is introduced. By defining a small area between two convex shapes, entry and exit points are created, enabling node placement based on the points reachable by both disjoint regions. The stages of constructing these passage nodes are discussed below, with stage numbers referencing the corresponding visualizations seen in Figure 3.2.

Stage 1-2

To create the passage nodes, obstacle pairs are first identified. Let $O = \{O_1, O_2, \ldots, O_n\}$ denote the set of known obstacles in the environment. Unique pairs (O_i, O_j) are generated where $i \neq j$, filtered based on their distance and mass properties. The pairs $P = \{(O_i, O_j) \mid d(O_i, O_j) \leq 2r\}$ consist of obstacles too closely spaced for the robot to pass between, with at least one obstacle in each pair being movable. For each pair $(O_i, O_j) \in P$, convex hulls A and B are constructed around the vertices of polygons O_i and O_j respectively. These hulls define the outer boundaries of the polygons, ensuring coverage of all vertices.

In Figure 3.2, the first two steps show a unique combination of two movable obstacles: a hexagon (O_1) and a square (O_2), with masses $m_1 = 20$ kg and $m_2 = 5$ kg, respectively. Both obstacles are already convex and below the fictional movability threshold of 30 kg. A fictional safety margin $r = 1$ meter is set to demonstrate that the minimum distance between the two shapes is below the threshold, $0.534 < 2r$.

Stage 3-4

Subsets $A' \subseteq A$ and $B' \subseteq B$ are identified by selecting the closest points between A and B. Specifically, each vertex of obstacle A is matched with the closest point on convex hull B, and vice versa. This process ensures A' and B' consists of points that establish the closest correspondence between the vertices of the two polygons, facilitating the creation of a new polygonal area C between O_i and O_j. Polygon C inherently defines two boundaries that encapsulate the area between the two polygons, serving as entry and exit points connecting the two areas.

In Figure 3.2, the closest correspondence points on the other polygon are shown as colored lines and points. Using these points, the outer boundary of the convex hull can be defined, where there are exactly two boundaries with points on both shapes, represented as the entry and exit boundaries.

Stage 5-6

A node is placed on both the entry and exit boundary and is shifted according to the masses of the obstacles. The starting point v_1 of the line within obstacle O_i or O_j is identified, along with the corresponding mass m_i or m_j. The total boundary length l is then divided by the total mass $m_i + m_j$ and multiplied by the weight of the mass of the obstacle where the node is positioned. The position of the node is adjusted to be further from the heavier obstacle and closer to the lighter one, with the interpolation length limited by the safety margin r. This is based on the following interpolation formula: $\mathrm{p} = (1-t)v_1 + tv_2$, where $t = \frac{m_j}{m_i + m_j}$ and the maximum interpolation length is r. This ensures that the node is placed proportionally closer to the lighter obstacle and further from the heavier one, while ensuring the placement does not exceed the safety margin r, balancing the placement for optimal navigation.

In Figure 3.2, the center of each boundary is shown, after which it is shifted away from the heavier obstacle (the hexagon O_i) and towards the lighter obstacle (the square O_j). The nodes are then labeled as passage nodes, indicating that they can serve as both entry and exit points.

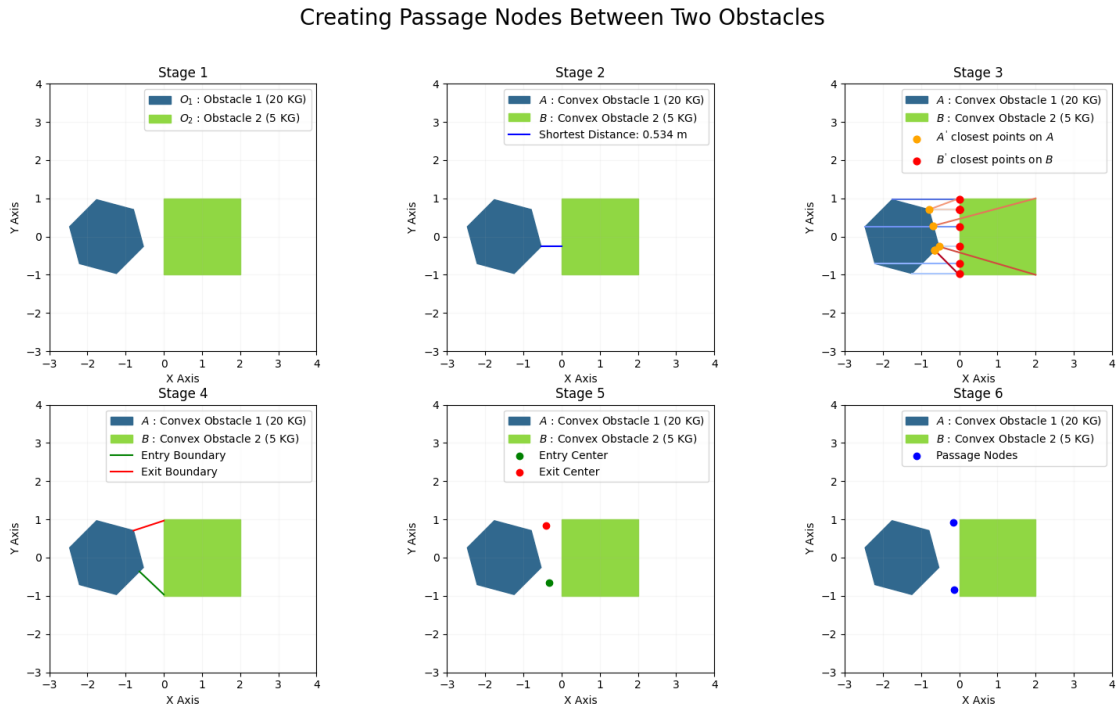


Figure 3.2: Example of a passage node construction between two convex polygons considering a safety margin of $r = 1$ meter.

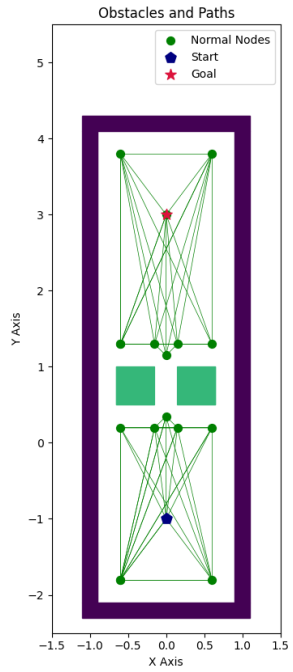


Figure 3.3: Normal Visibility Graph (VG).

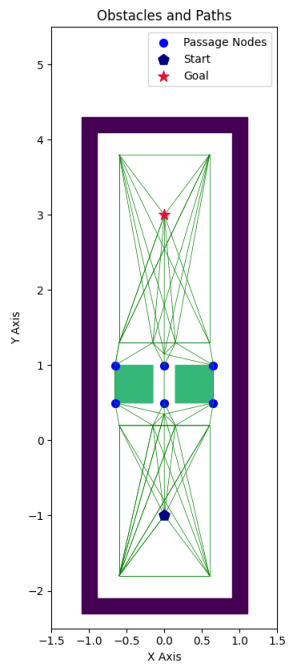


Figure 3.4: Semantic Visibility Graph (SVG)

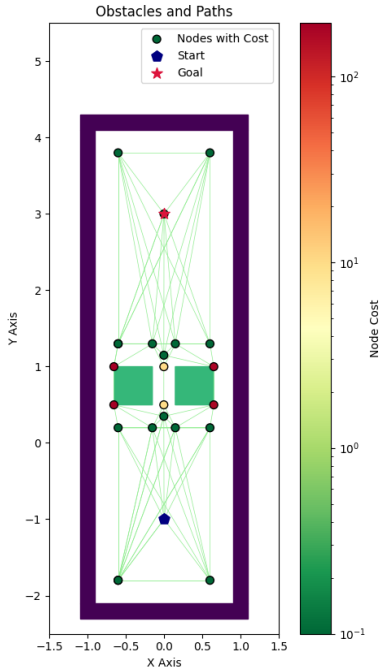


Figure 3.5: Weighted Semantic Visibility Graph (SVG).

Figure 3.6: A simple environment for comparison of Visibility Graph (VG) and Semantic Visibility Graph (SVG), including the weighted SVG with node costs.

Semantic Node Cost

The SVG algorithm extends Visibility Graphs to accommodate scenarios with movable obstacles through the strategic placement of nodes known as *passage nodes*. In this framework, edge costs represent the Euclidean distance between two nodes. Additionally, an optional node cost reflects the extra effort required to move an obstacle to access a passage node. This section outlines how these node costs are calculated.

Two subsets of nodes are defined: *free-space* nodes (V_{free}) and *passage* nodes (V_{passage}). Free-space nodes have a cost of zero, as no pushing action is required to access them. In contrast, passage nodes require a ”push” action for the robot to reach them. The cost for these nodes is determined based on the distance each obstacle must be moved to create the passage and the mass of the obstacles involved.

The boundary, illustrated in Figure 3.2 as entry and exit points i and j, defines the Euclidean distance from the passage point to the obstacles. For each passage node, the cost is computed using the distances from the passage point to the lighter and heavier obstacles, adjusted by their respective masses. Specifically, the cost is calculated by:

$$c_k = \begin{cases} \left(\max(0, 1 - \frac{d_i}{r}) \cdot m_i\right) + \left(\max(0, 1 - \frac{d_j}{r}) \cdot m_j\right) & \text{if } v_k \in V_{\text{passage}} \\ 0 & \text{if } v_k \in V_{\text{free}} \end{cases} \tag{3.1}$$

Here, d_i and d_j represent the distances from the passage point to the obstacles, while m_i and m_j are their respective masses. Points closer to heavier obstacles incur a higher cost due to the increased impact of the obstacles' weight. This reflects the greater effort required by the robot to move or navigate around these heavier obstacles.

This formulation calculates the cost based on the total effort required to move obstacles, taking into account both their distance from the passage point and their mass. Figure 3.6 illustrates a hallway scenario where the goal position is obstructed by two movable obstacles of equal weight. Figure 3.3 shows the standard Visibility Graph (VG), highlighting the disconnected regions of the graph. Figure 3.4 visualizes the passage nodes, while Figure 3.5 presents the weighted nodes, with color coding to indicate their associated costs.

Edge Case and Exceptions

Obstacles in indoor environments can vary widely in size and shape. Some obstacles have straightforward convex geometries, such as simple polygons, as illustrated in Figure 3.2. Others, like corner sofas, feature more complex, non-convex forms. The method proposed for identifying passage nodes is designed to handle these diverse scenarios effectively. Nonetheless, certain exceptions must be considered to understand the algorithm's robustness in different situations and edge cases.

The algorithm generates unique combinations of all known obstacles on the map, but there are cases where no passage can be created. For example, if both obstacles are classified as non-movable due to their mass, it is not possible to shift any obstacle to create a passage. Similarly, if the convex hulls of the obstacles overlap, the algorithm cannot place passage nodes effectively, resulting in no passage being created. When dealing with non-convex shapes, the algorithm approximates them as convex shapes. This conservative approach means that the algorithm assumes less space than is available.

Figure 3.7 illustrates the application of the algorithm to simple rectangular shapes, highlighting its effectiveness with basic convex geometries. Figure 3.8 shows various scenarios involving convex polygons. It includes cases where the shortest distance involves a node from each polygon, a node from one polygon, and a boundary from another, and where the closest points are on two boundaries. Finally, Figure 3.9 demonstrates how the algorithm handles non-convex shapes by treating them as convex, thereby providing conservative calculations for passage nodes.

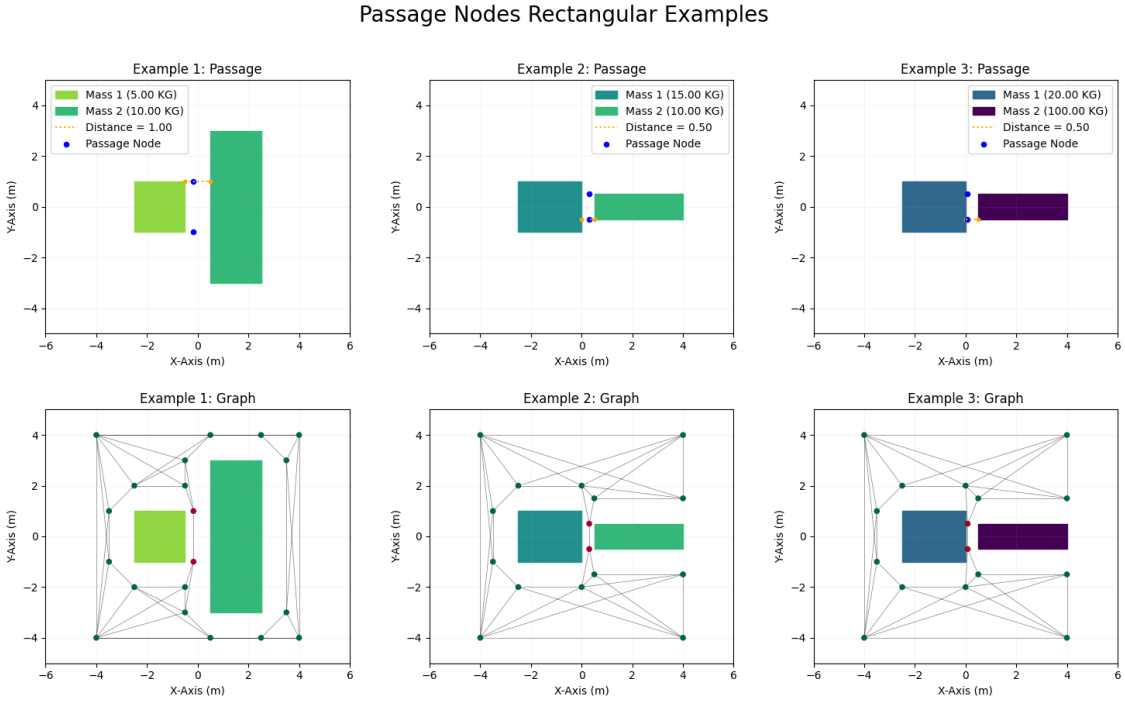


Figure 3.7: Examples of rectangular polygons and their corresponding passage nodes, showcasing three distinct examples: (1) Two movable obstacles with differing boundary lengths at the passage, (2) Two movable obstacles with similar boundary lengths at the passage, and (3) A single movable obstacle.

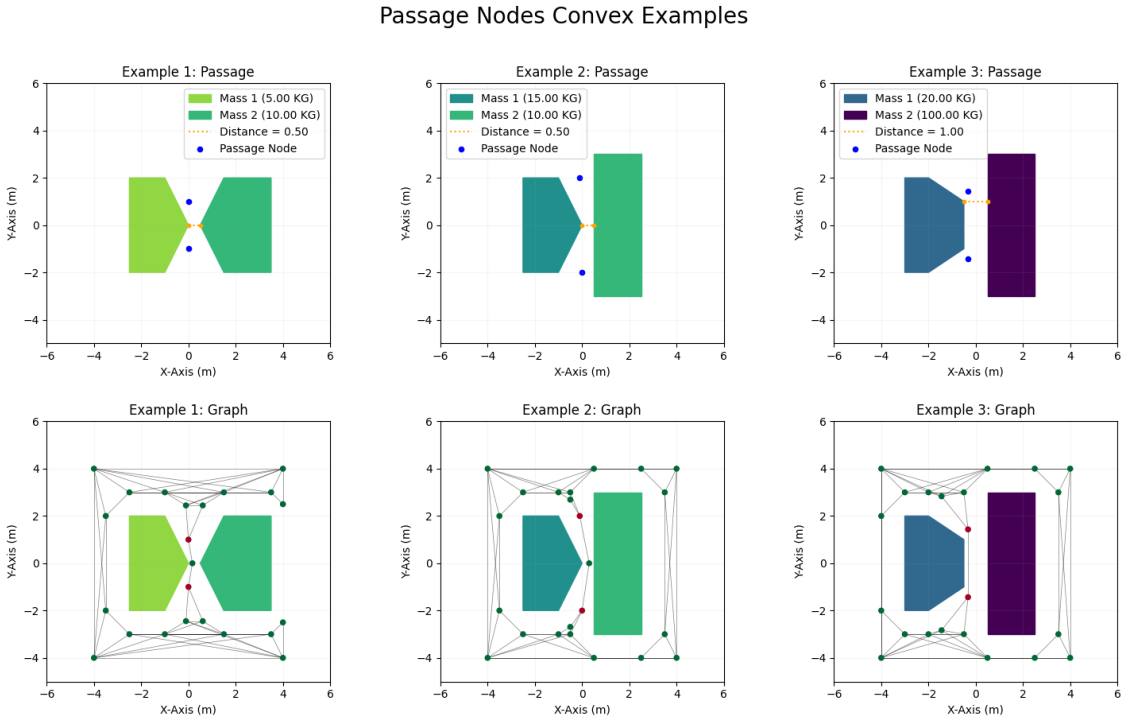


Figure 3.8: Examples of convex polygons and their corresponding passage nodes, illustrating three distinct examples: (1) Two movable obstacles with vertices at the passage, (2) Two movable obstacles with a vertex and an edge at the passage, and (3) One movable obstacle with two boundaries at the passage.

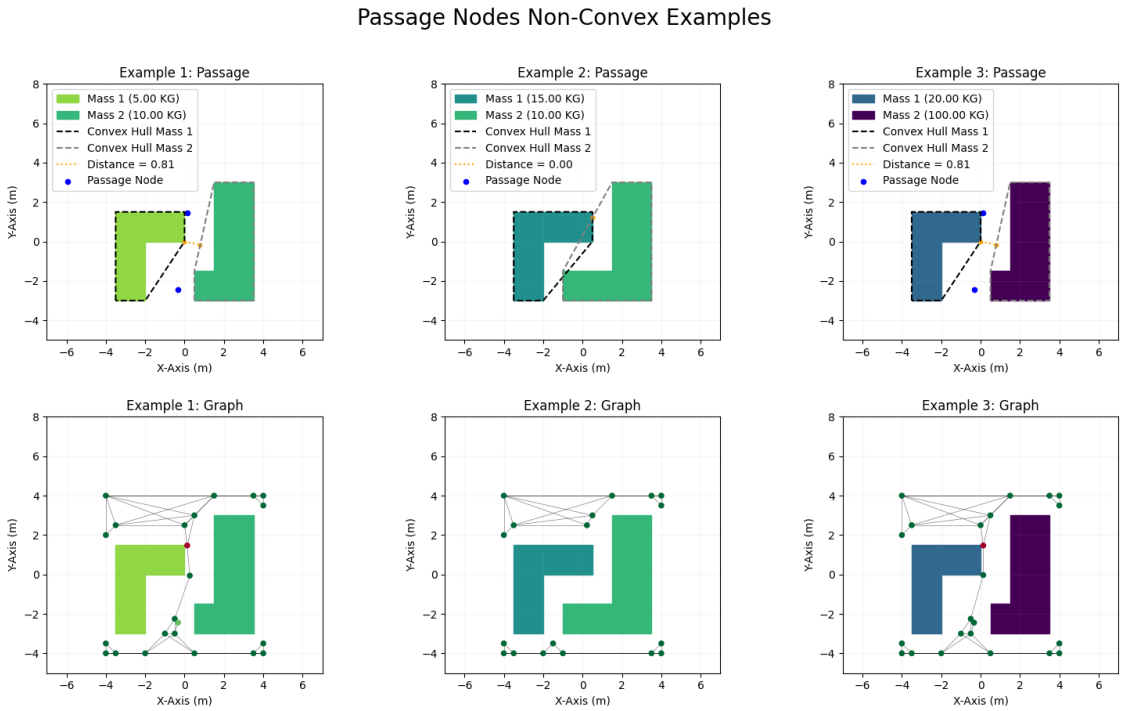
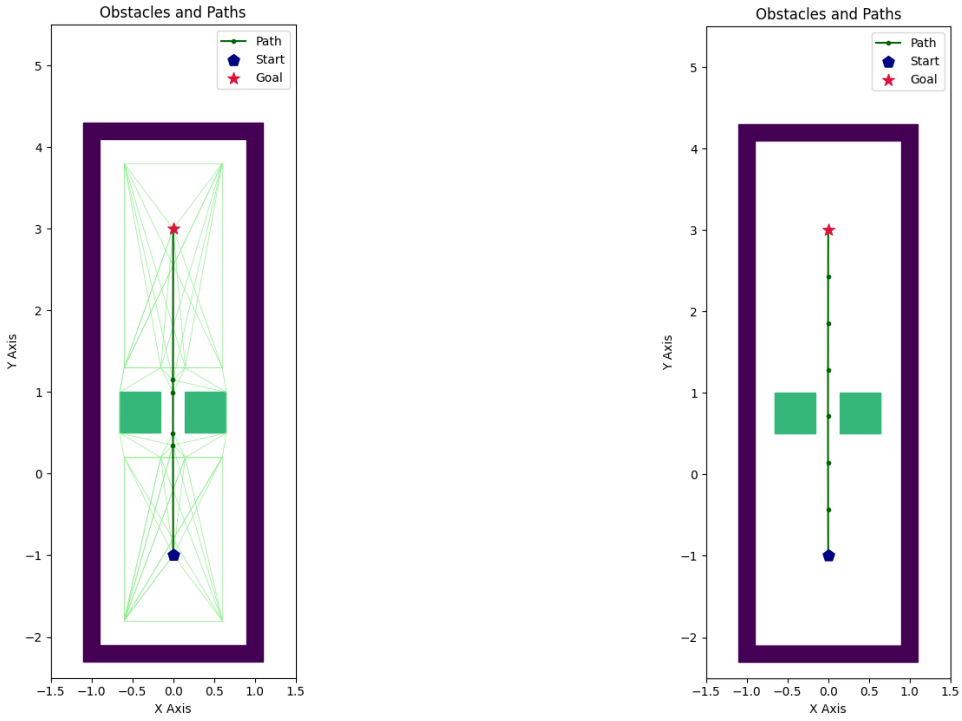


Figure 3.9: Examples of non-convex polygons and their corresponding passage nodes, showcasing three distinct cases: (1) Two movable obstacles with non-overlapping convex hulls, (2) Two movable obstacles with overlapping convex hulls, and (3) A single movable obstacle.



(a) Shortest path algorithm output.

(b) Interpolation on the shortest path output.

Figure 3.10: Defining the waypoints for the robot to follow based on the weighted graph of SVG.

3.2. Shortest Path

The Semantic Visibility Graph (SVG) creates a weighted graph where edge costs represent the distances between waypoints, and node costs reflect the effort required to move obstacles out of the way, as detailed in Section 3.1. This section describes the application of a shortest-path algorithm to find a path and the use of interpolation to ensure a consistent interval between waypoints.

In an SVG, all edge costs are greater than zero, and the graph is considered undirected. Visibility graphs inherently support multiple queries, enabling path planning from various start and end points, provided the environment remains unchanged. However, this flexibility is reduced when movable obstacles are present, as interactions with these obstacles alter the environment.

Dijkstra's algorithm is traditionally employed to efficiently find the shortest path on such graphs, computing routes from a single source node to other nodes with non-negative edge weights [31]. Among the variations of Dijkstra's algorithm, A* is notable for its use of heuristics to guide the search towards promising vertices [32]. The A* algorithm typically uses heuristics such as Euclidean or Manhattan distance to ensure admissibility (the heuristic never overestimates the true cost to reach the goal) and consistency (the heuristic is monotonically non-decreasing along the path).

Several adaptations of A* offer practical benefits for different scenarios. For instance, Bi-directional A* [63] performs simultaneous searches from both the start and goal nodes, potentially reducing search time by finding paths that converge midway. Anytime Repairing A* begins with a quick but suboptimal solution and progressively refines it over time [64]. Real-time Adaptive A* dynamically adjusts it's heuristic based on real-time feedback, making it useful for navigating environments with changing conditions or unexpected obstacles [65]. Dynamic A*, also known as D*, is designed for environments that are partially known or continuously evolving, proving particularly advantageous for robotic path planning [66].

Selecting the most suitable shortest path algorithm for the SVG-generated weighted graph depends on the graph's properties and the environment. When full environmental visibility is available, there are no dynamic entities, and the configuration space meets the criteria for admissibility and consistency, A* with Euclidean distance as the heuristic is most appropriate. This approach assumes a custom weight function where traversing an edge incurs the edge cost and visiting a node adds the node cost.

Figure 3.10a illustrates the result of applying the shortest path algorithm to the SVG-weighted graph from the previously discussed example. The figure demonstrates that the passage nodes are effectively utilized and that the direct line represents the shortest path from the start to the finish.

3.2.1. Waypoint Interpolation

This section describes the interpolation method used between generated waypoints to ensure a consistent interval between them. This approach standardizes the distance between waypoints, which is crucial for maintaining uniform behavior of the objective function across varying waypoint distances.

Interpolation estimates values within the range of known data points. Specifically, splines are used to create smooth curves that pass through these data points [67]. Among various spline types, linear splines connect points with straight lines, while quadratic and cubic splines use polynomial functions to provide smoother curves. Cubic splines, in particular, offer continuous first and second derivatives, balancing smoothness with complexity.

Despite the advantages of higher-order splines, such as better continuity in velocity and acceleration, they can introduce unwanted curvature that might intersect with obstacles. To avoid this, a first-order polynomial is employed for interpolation between waypoints. This approach uses straight lines with a default interval of $\alpha = 0.5$ meters, ensuring that waypoints are connected linearly and reducing the risk of intersecting obstacles.

Figure 3.10b illustrates how this interpolation method is applied to the shortest path algorithm's output. The default interval value helps maintain a consistent distance between waypoints, preventing nodes from being placed too close together and ensuring the path remains clear of obstacles.

3.3. MPPI with Contact-Force Minimization

The output of the shortest path algorithm, described in Section 3.2, which uses the weighted graph produced by the Semantic Visibility Graph algorithm (explained in Section 3.1), is a sequence of waypoints from the starting position to the goal. Using Model Predictive Path Integral (MPPI) as the local control strategy, the goal is to follow these waypoints effectively while minimizing the robot's push actions. This section details the constraints and objective function of MPPI to achieve precise waypoint following and reduce push actions by minimizing contact forces on the robot.

To understand the details of MPPI, it's essential to recap several key variables, as introduced in Section 2. MPPI involves sampling a total of K state trajectories Q_k, often referred to as rollouts, across a time horizon T. Each rollout Q_k represents a series of state vectors $\mathbf{x}_{t,k}$, meaning each rollout k has a series of prediction steps t over a total horizon of T. The state vector specifies the robot's position and orientation (x, y, θ). The derivative of the state vector, $\dot{\mathbf{x}}_{t,k}$, denotes the velocities associated with these state variables. In the context of MPPI, $\mathbf{v}_{t,k}$ signifies the action applied to the robot during each prediction step in each rollout. The action is based on velocity control for a holonomic robot, having the same degrees of freedom in configuration space as in state space.

3.3.1. Constraint Violation

The main goal of MPPI control is to minimize a cost function over a finite time horizon while satisfying a set of constraints. Instead of directly solving the optimization problem, MPPI samples various potential control trajectories and evaluates them using the cost function. As with many optimization problems, MPPI incorporates constraints that define the feasible region for solutions, addressing physical limitations, safety requirements, and other specific considerations.

In the proposed framework, SVG-MPPI, constraints are defined to ensure feasibility control inputs. Only hard constraints are applied, which must always be satisfied. The key constraints in SVG-MPPI are detailed in equations 3.2a, 3.2c, 3.2d, and 3.2b. The state transition constraint (3.2a) describes how the system evolves at each prediction step t based on its current state $\mathbf{x}_t$ and control inputs $\mathbf{v}_t$. Since MPPI is used in combination with Isaac Gym, the state transition is managed by the physics engine [40]. The state feasibility constraint (3.2b) requires the robot's state to remain within a defined feasible region throughout its operation. The control input constraint (3.2c) specifies the set of allowable actions for the robot, ensuring that velocities remain within the defined maximum and minimum limits for the holonomic robot. The initial state constraint (3.2d) ensures that the robot starts from a predetermined known pose.

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{v}_t) \quad , t = 0, 1, \ldots, T-1 \qquad \text{State Transition} \qquad (3.2a)$$

$$\mathbf{x}_t \in \mathcal{X} \quad , t = 0, 1, \ldots, T-1 \qquad \text{State Feasibility} \qquad (3.2b)$$

$$\mathbf{v}_t \in \mathcal{V} \quad , t = 0, 1, \ldots, T-1 \qquad \text{Control Input} \qquad (3.2c)$$

$$\mathbf{x}_0 = \mathbf{x}_{\text{init}} \qquad \text{Initial State} \qquad (3.2d)$$

3.3.2. Cost function

MPPI aims to minimize its objective function by evaluating the accumulated cost S_k for each sample k over a planning horizon T. As detailed in Equation 2.1 on page 8, S_k represents the cumulative cost across all prediction steps, discounted by a factor γ for each rollout k. The cost function $c(\mathbf{x}_{t,k}, \mathbf{v}_{t,k})$ assesses the state $\mathbf{x}_{t,k}$ and the action $\mathbf{v}_{t,k}$ applied at each prediction step t within rollout k. This section offers a detailed explanation of the cost function.

Each rollout is evaluated using the cost function at every prediction step t. IsaacGym's support for parallel computation allows these evaluations to be efficiently handled through matrix operations. Instead of calculating vectors for each prediction step t and rollout k individually, matrices are used to aggregate data. Specifically, for each prediction step t, a matrix is constructed where each row represents data from a different rollout. This approach enhances computational efficiency by leveraging parallel processing capabilities.

The total cost function is defined in Equation 3.3. This function distinguishes between stage costs and terminal costs. The stage cost is applied at every prediction step t whereas the terminal cost is applied solely at the final prediction step of the horizon.

$$\mathbf{c}(\mathbf{X}_t, \dot{\mathbf{X}}_t, \mathbf{V}_t, \mathbf{F}_t, \mathbf{P}) = \begin{cases} \mathbf{c}_{\text{terminal}}(\mathbf{X}_t, \dot{\mathbf{X}}_t, \mathbf{V}_t, \mathbf{F}_t, \mathbf{P}) & \text{, if } t = T-1 \\ \mathbf{c}_{\text{stage}}(\dot{\mathbf{X}}_t, \mathbf{V}_t, \mathbf{W}_{\text{control}}) & \text{, otherwise} \end{cases} \tag{3.3}$$

The stage cost function and terminal cost function are detailed in Equations 3.4a and 3.4b, respectively. Here, $\mathbf{X}_t$ represents the matrix of state vectors for all rollouts at prediction step t, $\dot{\mathbf{X}}_t$ is the matrix of velocity vectors, $\mathbf{V}_t$ denotes the matrix of control actions, and $\mathbf{F}_t$ is the matrix of contact force tensors provided by Isaac Gym for all actors in the environment. The matrix $\mathbf{P}$ contains all waypoints, determined by the shortest path algorithm and interpolated between these points.

$$\mathbf{c}_{\text{stage}}(\dot{\mathbf{X}}_t, \mathbf{V}_t) = \mathbf{c}_{\text{control}}(\dot{\mathbf{X}}_t, \mathbf{V}_t, \mathbf{W}_{\text{control}}) \qquad \text{Stage Cost} \tag{3.4a}$$

$$\begin{aligned} \mathbf{c}_{\text{terminal}}(\mathbf{X}_t, \dot{\mathbf{X}}_t, \mathbf{V}_t, \mathbf{F}_t, \mathbf{P}) = \mathbf{c}_{\text{control}}(\dot{\mathbf{X}}_t, \mathbf{V}_t, \mathbf{W}_{\text{control}}) \\ + w_{\text{distance}} \cdot \mathbf{c}_{\text{distance}}(\mathbf{X}_t, \mathbf{P}) \\ + w_{\text{progress}} \cdot \mathbf{c}_{\text{progress}}(\mathbf{X}_t, \mathbf{P}) \\ + w_{\text{rotation}} \cdot \mathbf{c}_{\text{rotation}}(\mathbf{X}_t, \mathbf{P}) \\ + w_{\text{force}} \cdot \mathbf{c}_{\text{force}}(\mathbf{F}_t) \end{aligned} \qquad \text{Terminal Cost} \tag{3.4b}$$

The individual components each address distinct penalty conditions to assess the robot's performance. By placing most of the cost terms in the terminal cost, longer horizons provide greater flexibility during the intermediate steps, as deviations are less penalized. The motivation is to enable the robot to deviate from the waypoints to find optimal trajectories around obstacles, thereby reducing push actions where possible. The weights assigned to each cost component determine its relative importance and are treated as hyperparameters. Table 3.1 provides explanations of the objectives for each component, supplemented with additional descriptions for clarity.

Table 3.1: Description of cost components for SVG-MPPI.

Symbol	Goal	Description
$\mathbf{c}_{\text{control}}$	Penalize differences between actions and velocities	Penalizes the difference between the desired velocity (action) and the current velocity at each time step along the horizon.
$\mathbf{c}_{\text{distance}}$	Minimize distance to the next waypoint	Penalizes the distance of the robot to the next desired waypoint position.
$\mathbf{c}_{\text{progress}}$	Favor progress over all the waypoints	Encourages the robot to advance along the path by rewarding progress towards waypoints.
$\mathbf{c}_{\text{rotation}}$	Alignment of the robot with target direction	Ensures that the robot is properly aligned with the direction of the next waypoints.
$\mathbf{c}_{\text{force}}$	Minimize contact forces with the environment	Penalizes the force exerted by the robot when it comes into contact with obstacles.

The total cost is derived from a weighted combination of individual components, resulting in a unitless value. Since each component operates on its own scale and range, the resulting penalties can vary significantly, leading to challenges in tuning due to potential imbalances. To address this, each component is normalized to ensure that its value falls within a standardized range of 0 to 1 before adjusting the weights. This normalization simplifies the subsequent tuning process. While each weight factor for the individual components is typically a scalar, the control weight consists of three distinct elements for each control input.

3.3.3. Cost Components

This section outlines the components of the cost function: distance cost, progress cost, control cost, rotation cost, and force cost. Each component is assigned a weight, which reflects its relative importance. These weights must be carefully tuned to achieve a balanced performance across different aspects of the robot's behavior.

Control Cost

To address jerky movements and oscillations around waypoints, a cost term is introduced to penalize significant deviations between the velocity command and the robot's current velocity. This cost is computed based on the absolute difference between these velocities. As detailed in Equation 3.5a, the difference is normalized by dividing by the maximum allowable velocity ($\mathbf{u}_{\text{max}}$), and then penalized quadratically using a weight matrix. The weight matrix $\mathbf{W}_{\text{control}}$ is diagonal, with each element corresponding to the weights for the velocity components in the x and y directions, as well as the angular velocity around the z-axis.

$$\mathbf{c}_{\text{control}}(\dot{\mathbf{X}}_t, \mathbf{V}_t, \mathbf{W}_{\text{control}}) = \left(\frac{|\dot{\mathbf{X}}_t - \mathbf{V}_t|}{\mathbf{u}_{\text{max}}} \right)^T \mathbf{W}_{\text{control}} \left(\frac{|\dot{\mathbf{X}}_t - \mathbf{V}_t|}{\mathbf{u}_{\text{max}}} \right) \tag{3.5a}$$

Distance Cost

The distance cost evaluates the importance of reaching the next waypoint. For the distance cost term, the Euclidean distance between the robot's current position and each waypoint is computed, as detailed in Equation 3.6b. In this distance matrix $\mathbf{D}_t$, each row corresponds to a different rollout, while each column represents the distance to a waypoint for that rollout.

For each rollout, the waypoint closest to the robot is identified by finding the waypoint $\mathbf{i}_t$ with the smallest distance in the corresponding row of $\mathbf{D}_t$, as described in Equation 3.6c. The next waypoint to target is determined based on the hyperparameter m. For SVG-MPPI, setting $m = 1$ means that the next waypoint to target is the one immediately following the closest waypoint. If the waypoints are too close together, m can be increased, but $\mathbf{i}_t + m$ must not exceed the total number of waypoints. Therefore, the constraint $1 < \mathbf{i}_t + m < |\mathbf{P}|$ must be satisfied, where $|\mathbf{P}|$ is the total number of waypoints.

The computed distances are then normalized by a factor α to ensure that the distance cost remains within the range of 0 to 1. For SVG-MPPI $\alpha = 0.5$, reflecting the interpolation interval used in shortest path calculations. This normalization prevents division by zero and maintains consistent scaling of distances. The resulting distance cost is given by Equation 3.6a.

$$\mathbf{c}_{\text{distance}}(\mathbf{X}_t, \mathbf{P}) = \frac{\mathbf{D}_t^{\mathbf{i}_t + m}}{\alpha} \qquad , 1 < \mathbf{i}_t + m < |\mathbf{P}| \tag{3.6a}$$

$$\mathbf{D}_t = \|\mathbf{X}_t - \mathbf{P}\|_2 \qquad \text{Distances to Waypoints} \tag{3.6b}$$

$$\mathbf{i}_t = \arg\min_j(\mathbf{D}_{t,j}) \qquad \text{Indices of Closest Waypoints} \tag{3.6c}$$

Progress Cost

The distance cost penalizes the deviation between the robot's position and the next waypoint relative to the closest waypoint. However, this approach might result in higher penalties for rollouts where the next waypoint has a higher index i, even if it is further along the track. To address this, a penalty term is introduced to account for the indices of the waypoints.

Similar to the distance cost, the indices $\mathbf{i}_t$ for each rollout are determined based on distances to all waypoints $\mathbf{D}$, and these indices are reused from the previous calculation. The relative difference r_t between the minimum and maximum waypoint indices is computed. If $r_t = 0$, it means all rollouts are targeting the same closest waypoint, so no additional penalty is applied.

If $r_t > 0$, indicating that different rollouts target different waypoints, a penalty is applied. This penalty is linearly scaled from 0 to 1, where the lowest indices receive a maximum penalty of 1, and the highest indices receive no additional penalty. This ensures that rollouts targeting waypoints further along the track, which are preferable, are not unfairly penalized.

The vector $\mathbf{i}_t$ contains the indices of the closest waypoints for each rollout, as determined by Equation 3.6c. The scalar r_t, representing the range of these indices, is computed using Equation 3.7b. The penalty term $\mathbf{c}_{\text{progress}}$, calculated by Equation 3.7a, adjusts the cost based on the waypoint indices. This approach ensures that rollouts that end further along the path are preferred over those that fall behind.

$$\mathbf{c}_{\text{progress}}(\mathbf{X}_t, \mathbf{P}) = 1 - \frac{\mathbf{i}_t - \min(\mathbf{i}_t)}{r_t} \tag{3.7a}$$

$$r_t = \max(\mathbf{i}_t) - \min(\mathbf{i}_t) \qquad \text{Range Of Indices} \tag{3.7b}$$

Rotation Cost

The rotation cost measures how well the robot's orientation aligns with the direction toward the next waypoint. This alignment offers two main advantages: it makes the robot's movement appear more natural by ensuring its front is oriented along the path, and it aids navigation through narrow passages by aligning the robot with the path as the width of a wheeled mobile robot is typically shorter compared to the length [68].

The target angle $\boldsymbol{\theta}_{\text{target}}$ is calculated using the atan2 function, which determines the angle between the robot's current orientation $\boldsymbol{\theta}_{\text{robot}}$ and the direction to the waypoint based on their relative positions, as shown in Equation 3.8b. The resulting angle is normalized by π to fit within the range $[0, 1]$, corresponding to the atan2 function's range of $[-\pi, \pi]$. The rotation cost $\mathbf{c}_{\text{rotation}}$ in Equation 3.8a quantifies the difference between the robot's current orientation and the computed target orientation.

$$\mathbf{c}_{\text{rotation}}(\mathbf{X}_t, \mathbf{P}) = \frac{|\boldsymbol{\theta}_{\text{target}} - \boldsymbol{\theta}_{\text{robot}}|}{\pi} \tag{3.8a}$$

$$\boldsymbol{\theta}_{\text{target}} = \text{atan}_2(\mathbf{P}_y - \mathbf{x}_t^y, \mathbf{P}_x - \mathbf{x}_t^x) \qquad \text{Target Angle} \tag{3.8b}$$

Force Cost

The force cost quantifies the interaction forces exerted on the robot by its environment. During interactions with obstacles, IsaacGym's physics engine computes the forces applied to both the robot's joints, which are described in the robot's URDF (Unified Robot Description Format), a standardized XML format that outlines a robot's physical structure, and the obstacles at each prediction step.

Rollouts that involve fewer push actions will accumulate less force over the prediction horizon T compared to those with more frequent push actions. As a result, rollouts with higher cumulative contact forces on the robot's joints incur a penalty. In Equation 3.9a, $\mathbf{c}_{\text{force}}$ represents the normalized force cost, where $\mathbf{f}_{\text{rob}}$ denotes the total cumulative force applied to the robot's joints over the prediction horizon T for each Rollout. The force matrix $\mathbf{F}_t$ at each prediction step t contains the forces f_{ij}^t applied to joint i at step t across all rollouts, with N being the total number of joints.

$$\mathbf{c}_{\text{force}}(\mathbf{F}_t) = \frac{\mathbf{f}_{\text{rob}}}{\max(\mathbf{f}_{\text{rob}}) + \epsilon} \tag{3.9a}$$

$$\mathbf{f}_{\text{rob}} = \sum_{t=0}^{T-1} \sum_{i=0}^{N-1} f_{ij}^t \qquad \text{Cumulative Force} \tag{3.9b}$$

The cumulative force $\mathbf{f}_{\text{rob}}$ is obtained by summing these forces across all joints and all prediction steps. The normalized force cost is calculated by dividing $\mathbf{f}_{\text{rob}}$ by the maximum cumulative force observed across all rollouts, with a small ϵ added to avoid division by zero. Although this penalty is considered part of the terminal cost, the forces are calculated and stored at each stage throughout the prediction horizon.

3.4. Movability Evaluation

The robot assesses the movability of objects based on their perceived mass distribution in the environment. Discrepancies between estimated and actual object properties, or incorrect object classification, can lead to inaccuracies. If an object behaves differently than expected, the planned trajectory using SVG-MPPI may become infeasible, necessitating replanning. This section outlines the conditions that trigger replanning.

Replanning is initiated when an object initially assumed to be movable is found to be non-movable. If an object fails to move as anticipated, its mass distribution is updated to the maximum threshold, reclassifying it as non-movable. This approach avoids recalibration based on observed behavior, which can be problematic due to difficulties in converting velocity feedback into accurate force and mass estimates. Replanning is considered only when the robot is approaching its final waypoint.

The conditions that trigger replanning are detailed in Table 3.2. If any of these conditions are met, a watchdog mechanism is activated. This mechanism uses a timer, t_{watchdog}, to monitor if a predefined threshold is exceeded. If t_{watchdog} surpasses τ_{replan} (set to 5000 milliseconds by default), replanning is triggered. Replanning involves taking a new snapshot of the environment, updating the movability data for the specific obstacle or obstacle class that did not behave as expected, and generating a new graph and path to the goal.

Specifically, obstacles initially assumed to be movable but found non-movable will have their mass distribution updated to the maximum threshold, reflecting their true non-movability. This update affects both the global planning graph and the MPPI environmental data. If none of the conditions are triggered, t_{watchdog} is reset to zero, pausing the replanning process until a condition activates it again.

Table 3.2: Conditions for Replanning, their Descriptions, and Default Values.

Condition	Description	Default Value
Velocities are too close to zero	Check if the robot's joint velocities are very low, which may indicate the robot is stuck or facing significant resistance. If the magnitude of joint velocities $\dot{\mathbf{x}}_{\text{rob}}$ falls below the threshold, it suggests ineffective movement.	$\epsilon = 0.1$
Desired velocity deviation from actual velocity	Assess whether actual joint velocities significantly deviate from the desired velocities. A deviation is significant if it exceeds λ percent of the desired velocities. If the difference between actual velocities $\dot{\mathbf{x}}_{\text{rob}}$ and desired velocities $\mathbf{v}_{\text{desired}}$ exceeds λ times the magnitude of the desired velocities, replanning is triggered.	$\lambda = 0.75$
The robot is slipping	Evaluates whether the robot is slipping, indicated by a stable position despite significant joint velocities. The robot is considered to be slipping if its position $\mathbf{x}_{\text{rob}}$ remains relatively constant while its joint velocities $\dot{\mathbf{x}}_{\text{rob}}$ exceed μ.	$\mu = 0.1$

4

Experimental Setup

This section details the experimental setup used to validate and benchmark the proposed solution, covering both qualitative and quantitative aspects. The qualitative experiments demonstrate the algorithm's functionality and its ability to navigate to a blocked goal position in both simulated and real-world environments. Meanwhile, the quantitative experiments compare the proposed method with other approaches, emphasizing the benefits of quantified movability for enhanced navigation.

The discussion starts with an overview of the experimental framework, including the hardware used and the SVG-MPPI hyperparameters. It then provides a comprehensive explanation of the qualitative and quantitative experiments conducted.

4.1. Experimental Framework

The experiments utilize the Dingo-O robot, an indoor mobile robot from Clearpath Robotics, known for its omnidirectional wheels that enable movement in any direction without changing orientation, making it holonomic [1]. Although the Dingo-O lacks onboard sensors for navigation or mapping, it features a ROS interface for precise control over its velocity, including linear velocities in the x and y directions and angular velocity around the z-axis.

In the simulation environment, the Dingo-O is represented using the URDF model. For real-world experiments, the actual physical robot is used. Both in simulation and the real world, obstacles are limited to rectangular shapes due to IsaacGym's support for only these convex customizable objects. For perception tasks such as object localization and semantic labeling, IsaacGym provides tensor data. Real-world experiments are conducted in a laboratory equipped with a Vicon high-precision tracking system [2].

A summary of the experimental parameters is provided in Table 3 in Appendix C. The maximum push mass is set to 30 kg, ensuring safe operation without risking damage to the robot or obstacles. A safety margin of $r = 0.3$ m, based on the robot's width of 517 mm, is applied to avoid collisions and ensure smooth navigation between obstacles.

For waypoint navigation, the interpolation interval is set to $\alpha = 0.5$ m. This interval provides a balance between computational efficiency and path smoothness. Replanning is triggered if the condition for replanning is met for $\tau_{\text{replan}} = 30$ seconds, indicating that the robot needs to update its path and environmental data. In the MPPI (Model Predictive Path Integral) control configuration, the simulation uses a time step of $dt = 0.08$ seconds with a prediction horizon of 25 steps, resulting in a total prediction time of 2 seconds. These settings are optimized to balance responsiveness and planning accuracy.

[1] https://clearpathrobotics.com/dingo-indoor-mobile-robot/
[2] https://www.vicon.com/

Metrics

The following metrics are used to evaluate the performance of the solution in the quantified experiments:

Table 4.1: Metrics for evaluating the performance in the quantified experiments.

Metric	Unit	Description
Path-finding Success Rate	Ratio (0-1)	Proportion of successful attempts by the planner in finding a viable path from the start to the goal position. Calculated as the ratio of successful trials to the total number of trials.
Reaching Goal Success Rate	Ratio (0-1)	Proportion of successful attempts by the robot in reaching the goal position. Calculated as the ratio of successful runs to the total number of simulations.
Planner Computation Time	Milliseconds	Time required by the planner to generate a path solution.
Goal Achievement Time	Seconds	Total duration taken by the robot to reach the goal position from the start.
Trajectory Contact Force	Newtons	Contact force exerted by the robot while navigating over the entire trajectory, with higher values indicating greater effort and lower efficiency.

4.2. Qualitative Experiments

The qualitative experiments aim to showcase the capabilities of the SVG-MPPI algorithm in navigating to a blocked goal position while effectively managing both stationary and movable obstacles. This section is divided into two parts: a simulated environment demonstration and a real-world application.

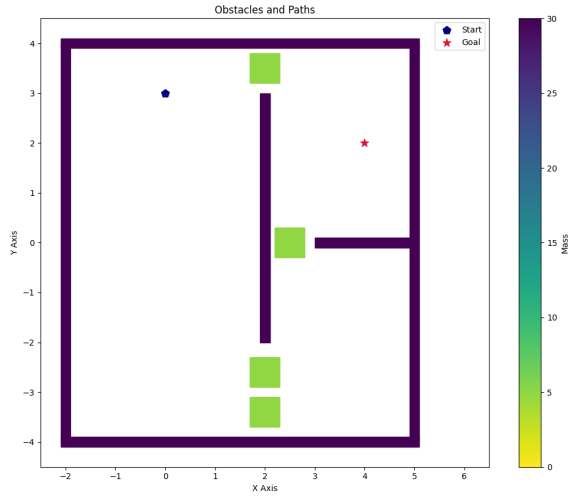
4.2.1. Setup A: Simulated Demonstration

Figure 4.1 presents a simple room setup where various obstacles obstruct the path to the goal position. In this scenario, it becomes apparent that moving the top obstacle provides the shortest route to the goal, which the robot is designed to prioritize. Additionally, this experiment will demonstrate the algorithm's adaptive behavior when the mass of the top obstacle is increased, rendering it immovable. When the obstacle cannot be pushed, the robot recognizes the need to reroute and uses the lower movable obstacles to find an alternative path to the goal. This highlights the algorithm's flexibility and responsiveness to wrong estimations of obstacle movability in the environment.

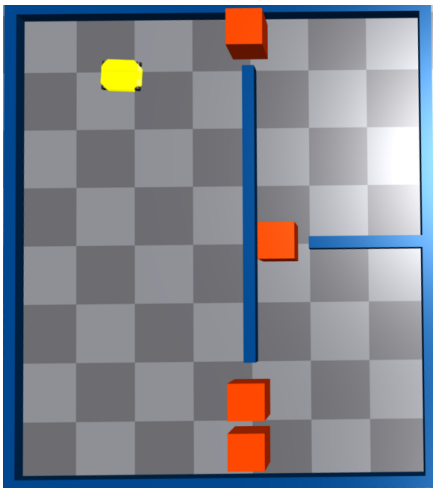
4.2.2. Setup B: Real World Demonstration

The real-world experiment illustrates the practical application of the SVG-MPPI algorithm on a physical robot. Using the Vicon motion capture system, which provides precise positioning and orientation data, the experiment replicates the perception pipeline used in simulations. Figure 4.2 depicts the layout of obstacles in the environment, effectively blocking a ”hallway” the robot needs to navigate through. In this setup, three obstacles (A, B, and C) are designated as movable, with masses of 25 kg, 20 kg, and 5 kg respectively. The boundaries of the hallway are treated as immovable walls, which the robot must work around.

The robot's task is to reach the other side of the hallway by moving the obstacles to create a passage. This experiment demonstrates the algorithm's ability to define and execute a viable path in a real-world setting, considering the physical constraints and properties of the obstacles. The results section will provide a detailed account of how the algorithm plans and adjusts the robot's path, showcasing its practical effectiveness and robustness.

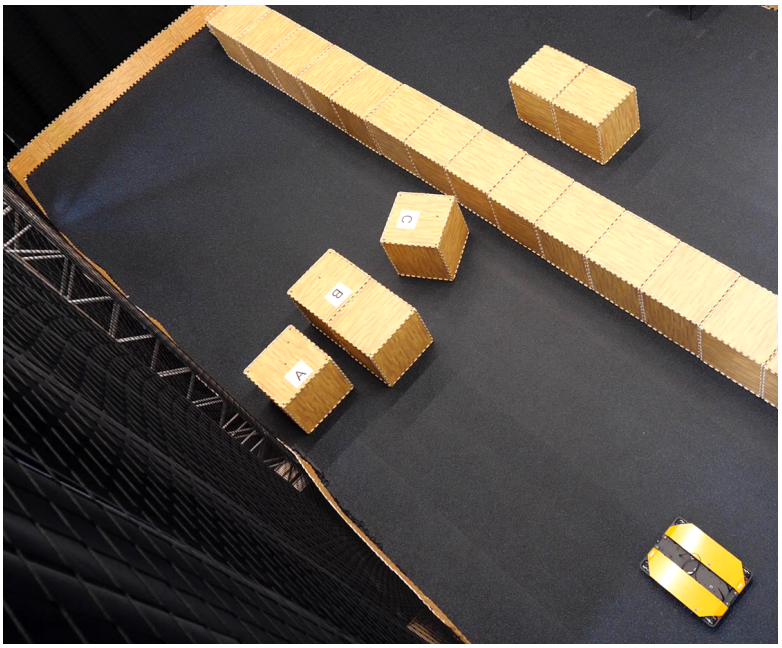


(a) Map with movable obstacles blocking the goal position for the robot.

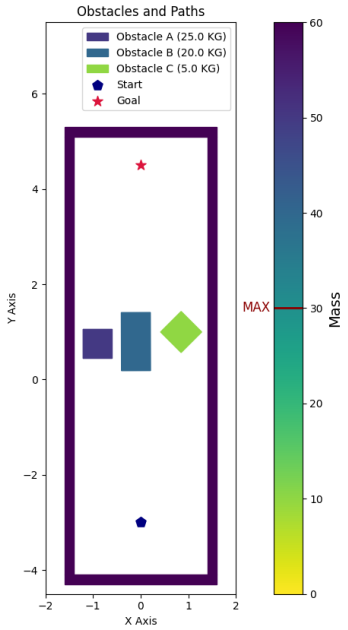


(b) The simulated IsaacGym environment with the obstacles.

Figure 4.1: Simulation with movable obstacles blocking the goal position for the robot.



(a) Vicon camera view of the real-world environment.



(b) Map of the real-world environment with obstacles.

Figure 4.2: Real-world demonstration with three obstacles Real-world demo of the robot navigating a hallway with three obstacles (A, B, and C) having masses of 25 kg, 20 kg, and 5 kg, respectively. The robot is tasked to move from one side of the hallway to the other.

4.3. Quantitative Experiments

This experiment assesses the performance of the SVG-MPPI solution in various simulated environments. Two distinct setups are used: one with a room featuring organized obstacles blocking the goal position, and another with a room cluttered randomly to obstruct the goal. This section outlines the comparison with alternative solutions and the rationale behind their selection.

To the authors' knowledge, this approach is novel in its continuous quantification of movability and its integration into both path planning and local control methods. As such, there are no existing off-the-shelf solutions or published statistics for direct comparison. Instead, comparisons are made with two well-established path-planning methods from the literature that have been adapted for this study.

For the quantitative experiments, two alternative path-planning methods are introduced alongside the proposed SVG solution. The first method is the standard Visibility Graph (VG) algorithm, which does not account for obstacle movability and maintains a safe distance from all obstacles, as detailed in Chapter 2. The second method is a more advanced, sampling-based technique using the Rapidly Exploring Random Tree (RRT) algorithm. RRT is widely used in the literature, as shown in Table 1.1 on Page 4, making it a suitable benchmark. In this study, RRT is modified to incorporate binary knowledge of movability, allowing it to sample near movable obstacles without a safety margin while avoiding stationary obstacles. However, RRT does not distinguish between obstacles that are easy or difficult to move; this modification is detailed in Appendix B.

Three path planning methods are compared: one without knowledge of movability (VG), one with binary knowledge of movability (modified RRT), and the proposed solution with a continuous scale of movability (SVG). All three planners use the same local control method, MPPI, which interprets obstacle masses according to the planner's movability knowledge. This means that while the planners operate in the same environment, the interpretation of obstacle movability differs depending on the planner-runner combination.

Each setup is executed 50 times in simulation, with obstacle masses uniformly randomly assigned between 4 and 36 kg and a mass threshold set at 30 kg, similar to the qualitative experiments. The SVG method is expected to outperform other path-planning approaches by strategically placing nodes within the map. MPPI will benefit from the continuous movability scale, as contact force minimization allows it to better adapt its path based on these varying masses.

4.3.1. Setup 1: Organized Environment

The first setup features a room measuring 4 meters by 12 meters, populated with fixed-position obstacles that are randomly rotated. These random rotations create varying boundaries, leading to different gaps between obstacles, which in turn generate preferred routes where the gaps are larger. Each obstacle is a fixed-size square with a randomly assigned mass. After constructing the room, each path planner attempts to find a path from the start to the goal. If a path is found, it is executed using MPPI while adhering to the specific movability properties of the planner. Performance metrics are collected and stored for later analysis. Figure 4.3 shows an example of a random room for this setup, illustrating different interpretations of movability: the first figure shows no movability knowledge, the second shows binary movability, and the third depicts continuous movability, as used by VG, RRT, and SVG respectively.

4.3.2. Setup 2: Cluttered Environment

The second setup simulates a more realistic, cluttered environment with obstacles that vary in position, rotation, size, and mass. This setup uses a slightly smaller room, measuring 4 meters by 8 meters, to reduce runtime while increasing complexity. Approximately 20% of the room is occupied by obstacles with uniformly random masses between 4 and 36 kg. To further increase complexity, an additional 5% of stationary obstacles is added, bringing the total occupied space to 25% within a y-axis range of -2 to 2 meters, thus increasing density. The obstacles vary in width and length, uniformly distributed between 0.25 meters and 1.25 meters, ensuring they are sufficiently large to be effectively pushed aside by the robot. Figure 4.4 provides an example of a random room for this setup, illustrating different interpretations of movability: the first figure shows no movability knowledge, the second shows binary movability, and the third depicts continuous movability, as used by VG, RRT, and SVG respectively.

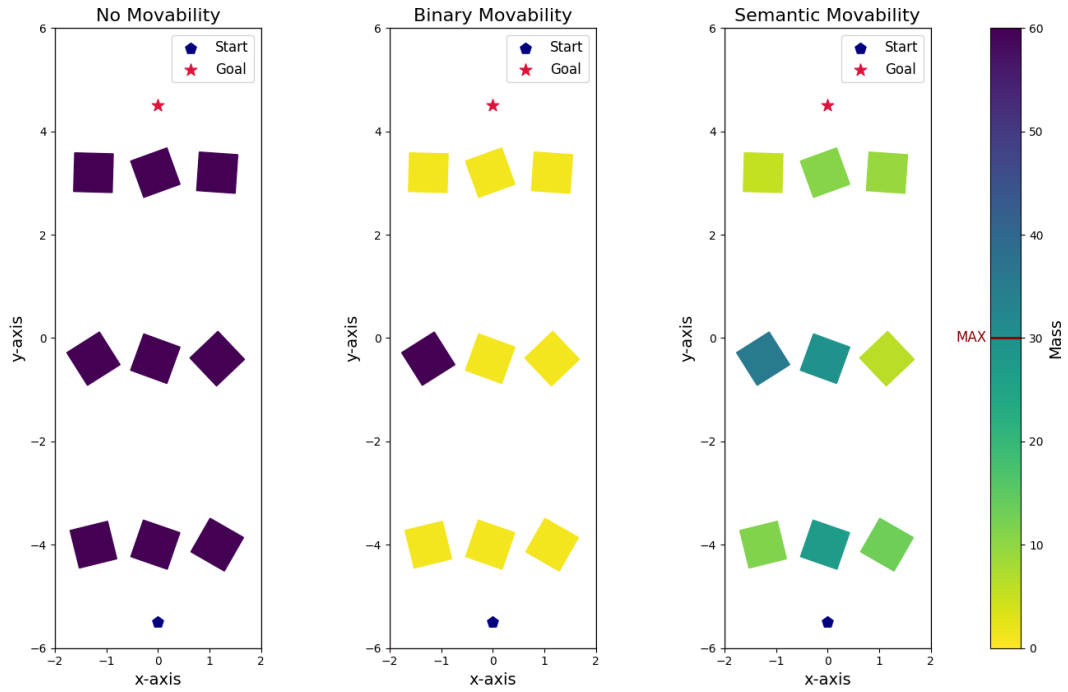


Figure 4.3: Setup 1: Two examples of a grid layout with varying obstacle masses and rotation.

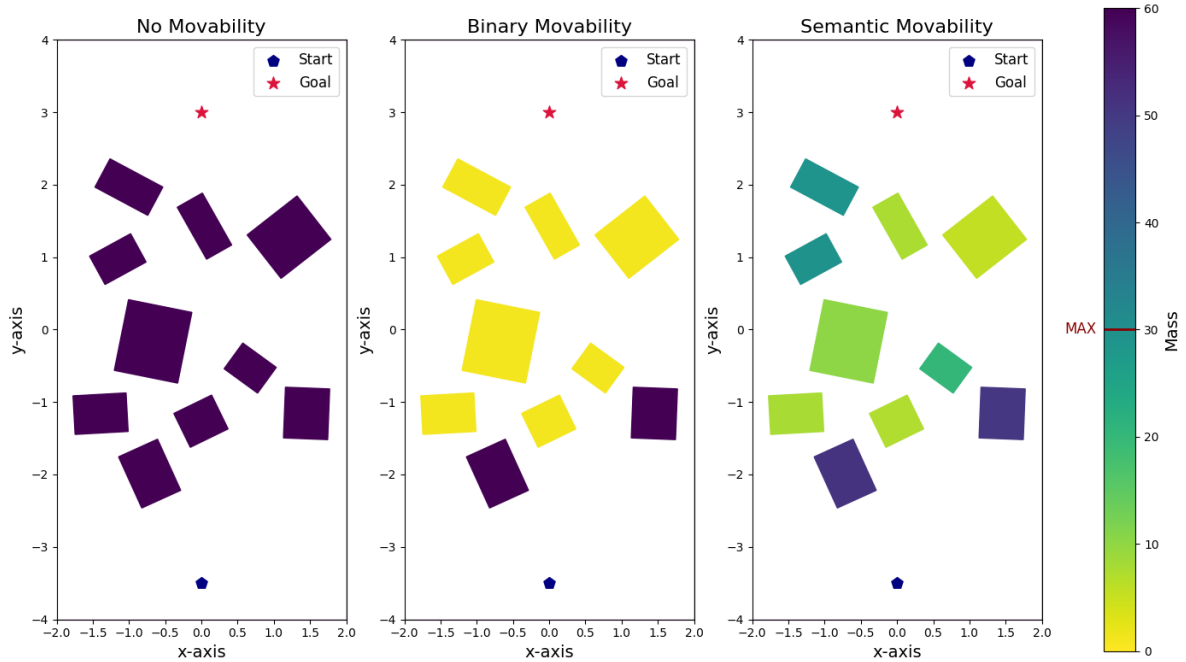


Figure 4.4: Setup 2: Two examples of a random layout with varying obstacle mass, position, and rotation.

5

Results

This chapter details the results from experiments designed to evaluate the performance and effectiveness of the proposed solution. The chapter begins with a discussion of qualitative experiments, which demonstrate how the SVG-MPPI algorithm operates in practice. This section focuses on the practical application of the algorithm without delving into the quantitative metrics. Following this, the chapter transitions to the quantitative experiments, which benchmark the solution using the previously discussed performance metrics.

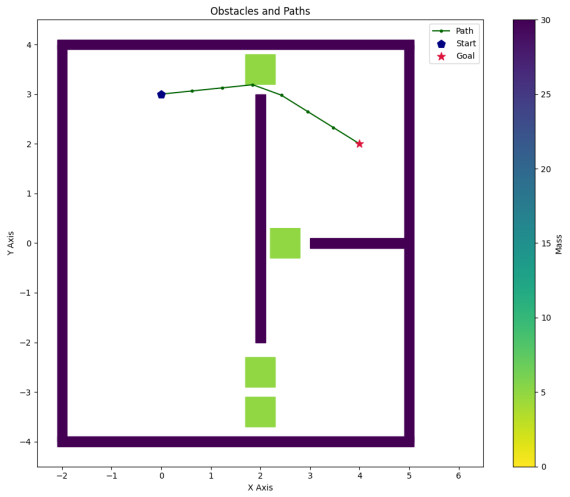
5.1. Qualitative Experiments

The initial experiment conducted in simulation, as shown in Figure 5.1, demonstrates how the algorithm finds and follows a path to the goal position using a local control strategy. In this setup, the robot pushes an obstacle slightly to create a passage, allowing it to move toward the goal. Tuning the parameters for this experiment was challenging due to the inherent contradiction of minimizing contact forces while still needing to push obstacles. Balancing these requirements was difficult because the robot needed to apply enough force to move obstacles but also had to avoid excessive contact forces to prevent damage or inefficiency. Additionally, the simulation environment allowed for precise trajectory following with minimal overshoot, which is more difficult to achieve in the real world.

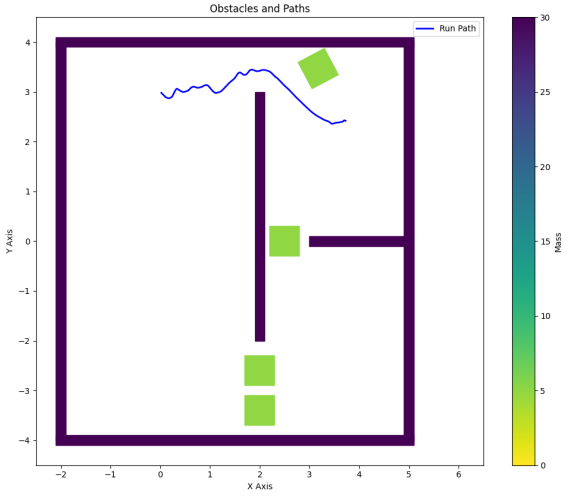
Another issue stemmed from the noisy force data provided by IsaacGym. This noise particularly affected the rear joints of the robot, causing unnecessary pushing actions when the robot turned and complicating the parameter tuning process. Despite these difficulties, the robot was still able to reach the goal position in the simulation, demonstrating the algorithm's effectiveness in a controlled environment.

Figure 5.2 shows a modified scenario within the same environment. Here, when the robot encounters the first obstacle, the pushing action fails, and the robot must replan its path. The obstacle, initially considered movable, is reclassified as non-movable, leading the robot to find a new route. This adjustment requires the robot to first navigate to the upper obstacle and then reroute through an alternative path to reach the goal. The figure illustrates the slight displacement of the upper obstacle and the robot's adjusted path. During this replanning, the MPPI rollouts initially struggled to find the best solution, resulting in some uncertainty in the robot's movements.

The algorithm's performance in the real world is shown in Figure 5.3. In this experiment, the Vicon system accurately tracks obstacle positions and orientations, effectively simulating the perception pipeline. Additionally, the graph and path associated with this behavior can be found in Appendix D. The robot's goal is to move to the other side of the hallway by relocating an obstacle to create a passage. This real-world test confirms that, despite the challenges of translating simulation results to practical applications, the algorithm successfully guides the robot to the goal position.

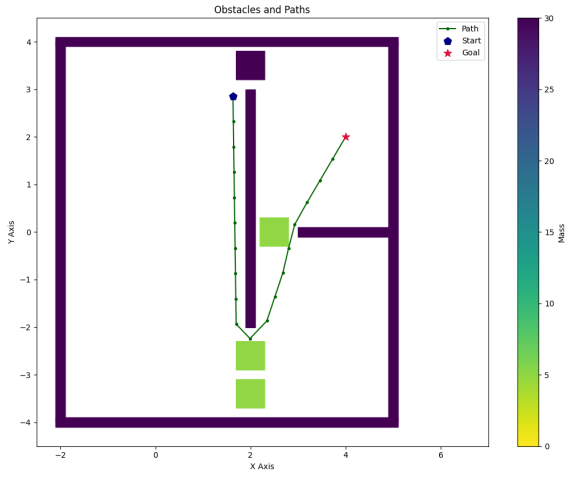


(a) The path calculated by SVG to reach the goal.

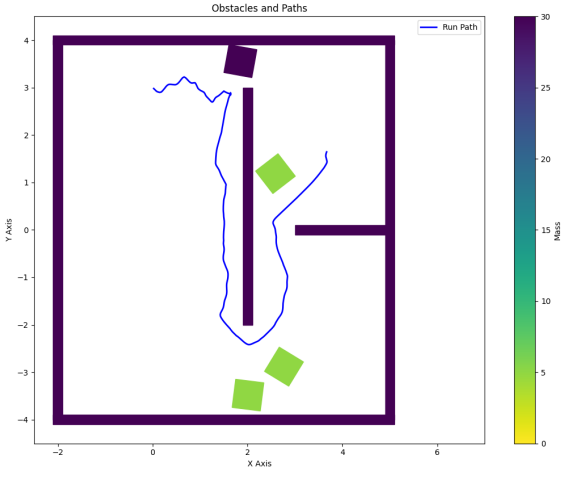


(b) The path executed by the robot to reach the goal.

Figure 5.1: Example of the robot reaching the goal with estimations being correct.



(a) Path replanning when the original path fails.



(b) The path executed by the robot including the replanning.

Figure 5.2: Example of the robot reaching the goal with evaluation triggering a replanning sequence.

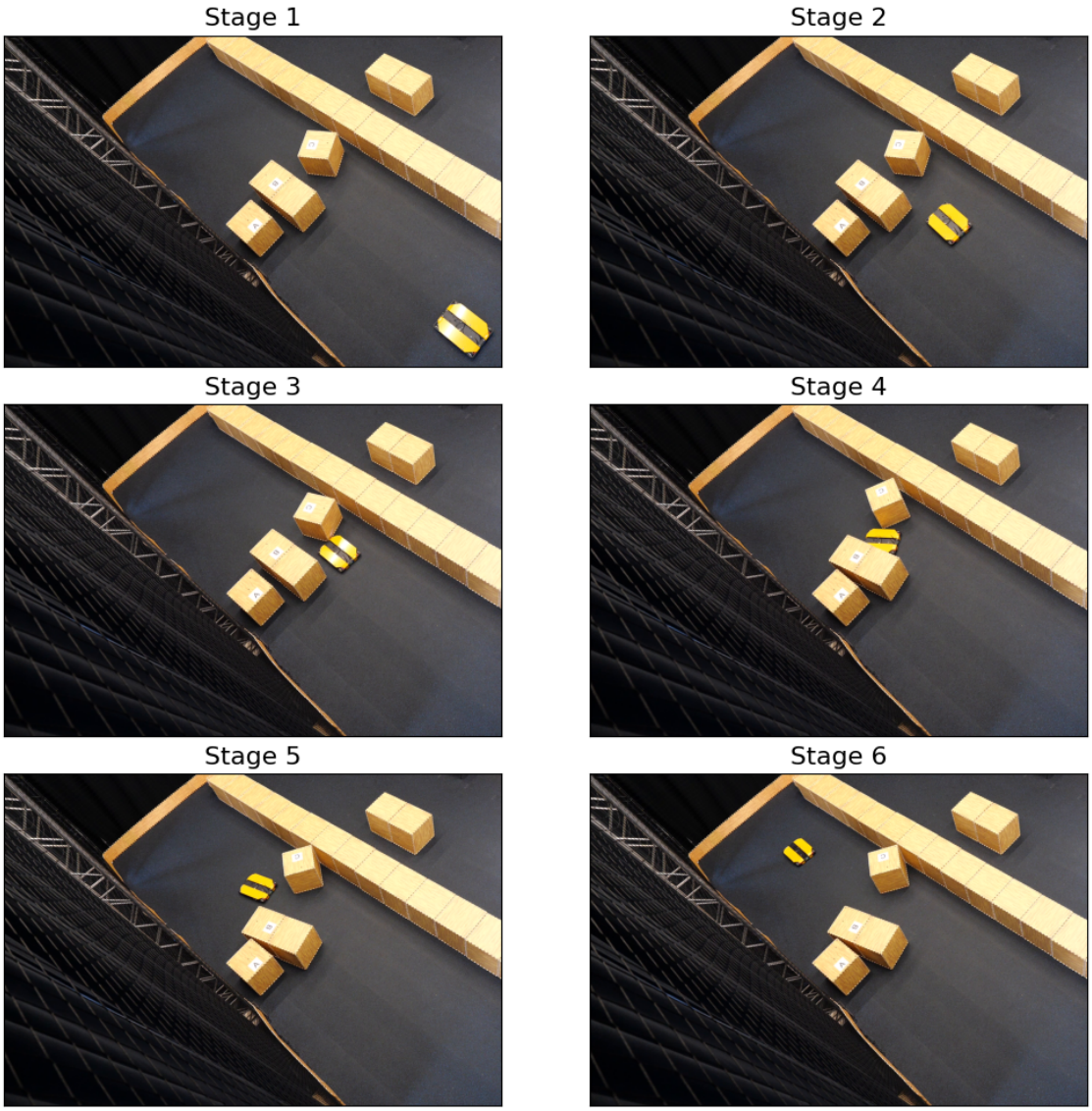


Figure 5.3: Real-world demonstration of the robot navigating a hallway, depicted in six stages. Stage 1 shows the initial position of the robot, and Stage 6 shows the goal position. In the intermediate stages, the robot moves towards the right side of the barrier, which is a lighter obstacle, and successfully pushes it to clear a path to the goal.

5.2. Quantitative Experiments

Table 5.1 presents the success percentages for path planning and execution by the planners—VG, RRT, and SVG—in Setup 1 and Setup 2. Table 5.2 shows the mean times for each planner to generate waypoints and execute paths. Due to the randomized nature of the room setups, the analysis focuses on the mean performance and the reliability of these estimates. Therefore, we use the standard error (SE) rather than the standard deviation (SD) as SE provides a more accurate measure of how well the sample mean reflects the true population mean [69]. Additional visual examples of the solutions for each setup are provided in Appendix D, displaying the weighted graphs and shortest paths of a single run for each planner in Setup 1 and Setup 2.

Success Ratio in Path Planning and Execution

In Setup 1, VG had a *0%* success rate in both path planning and execution due to its inability to navigate around fixed barriers. RRT achieved a path planning success rate of *95.65%* and an execution success rate of *65.22%*, indicating effective path-finding but challenges during execution, likely due to the variability inherent in its sampling-based approach. SVG achieved a *100%* path planning success rate and a *69.57%* execution success rate, showing reliable path-finding but occasional difficulties navigating large obstacles.

In Setup 2, with randomly placed obstacles, VG improved slightly with a *6.45%* success rate in both planning and execution, as clear paths are infrequent. RRT's success rates were *77.42%* for planning and *35.48%* for execution, indicating better path-finding but ongoing challenges with execution due to less optimal node placement. SVG showed significant improvement with a *96.77%* path planning success rate and a *77.42%* execution success rate, reflecting its adaptability to the less constrained environment of Setup 2.

Planning and Execution Latency

In Setup 1, VG's data are not available due to its failure to find a path. RRT had a mean planning time of *11.45 sec* with a high standard error of *10.80 sec*, suggesting significant variability in its sampling-based approach. SVG had a mean planning time of *0.14 sec* with a very low standard error of *0.002 sec*, reflecting consistent performance due to its deterministic approach to node placement. The difficulties both RRT and SVG experience in executing the paths are likely due to large obstacles that prevent effective sampling around objects, leading to either continuous pushing or collisions by the robot.

In Setup 2, VG demonstrated efficient planning with a mean time of *0.12 sec* and a standard error of *0.05 sec*. VG's execution time was *17.30 sec* with a standard error of *0.74 sec*, indicating straightforward paths due to the smaller room size. RRT's mean planning time improved to *0.82 sec* with a standard error of *0.42 sec*, reflecting better path-finding as the randomized environment often presents multiple routing options to the goal. However, RRT's execution time was higher, likely due to navigating close to heavy obstacles, which can create new boundaries and make the RRT path infeasible. SVG maintained a low mean planning time of *0.14 sec* with a standard error of *0.006 sec*, while its execution time remained efficient, benefiting from better passage node placements that avoid heavy obstacles.

Trajectory Contact Force

Figure 5.4 shows the cumulative contact force exerted by the robot during trajectory execution, measured in Newtons. The force is recorded at a rate of 25 times per second. This high sampling frequency means that even minor push actions are captured frequently, which can result in high cumulative force values. In Setup 1, VG had a significantly lower average cumulative force of *308.66 N* with a standard error of *180.06 N* because this planning method focuses solely on avoidance. In comparison, RRT applied an average cumulative force of *557,900.67 N* with a standard error of *104,596.54 N*, while SVG applied *416,354.08 N* with a standard error of *108,355.60 N*. For Setup 2, VG again showed a notably lower average cumulative force of *308.66 N* with a standard error of *180.06 N*. RRT's average cumulative force was *297,360.47 N* with a standard error of *125,251.49 N*, and SVG's was *123,463.09 N* with a standard error of *30,835.55 N*. RRT's consistently higher total trajectory force demonstrates that integrating continuous movability is beneficial for node placement and contact force reduction. While higher cumulative forces generally indicate more effort, this metric may be suboptimal as it aggregates force measurements without distinguishing the effectiveness or impact of the push actions. The frequent measurement can amplify the apparent force, as even small push actions contribute significantly to the cumulative total.

Table 5.1: Success Percentages by Setup and Planner

Setup	Planner	Path Planning Success (%)	Execution to Goal Success (%)
1	VG	0.00	0.00
	RRT	95.65	65.22
	SVG	100.00	69.57
2	VG	6.45	6.45
	RRT	77.42	35.48
	SVG	96.77	77.42

Table 5.2: Timing Metrics by Setup and Planner

Setup	Planner	Planner Time (s)		Runner Time (s)	
		Mean	SE	Mean	SE
1	VG	-	-	-	-
	RRT	11.45	10.80	106.16	16.15
	SVG	0.14	0.002	109.91	19.65
2	VG	0.12	0.05	17.30	0.74
	RRT	0.82	0.42	62.21	16.53
	SVG	0.14	0.006	43.57	5.74

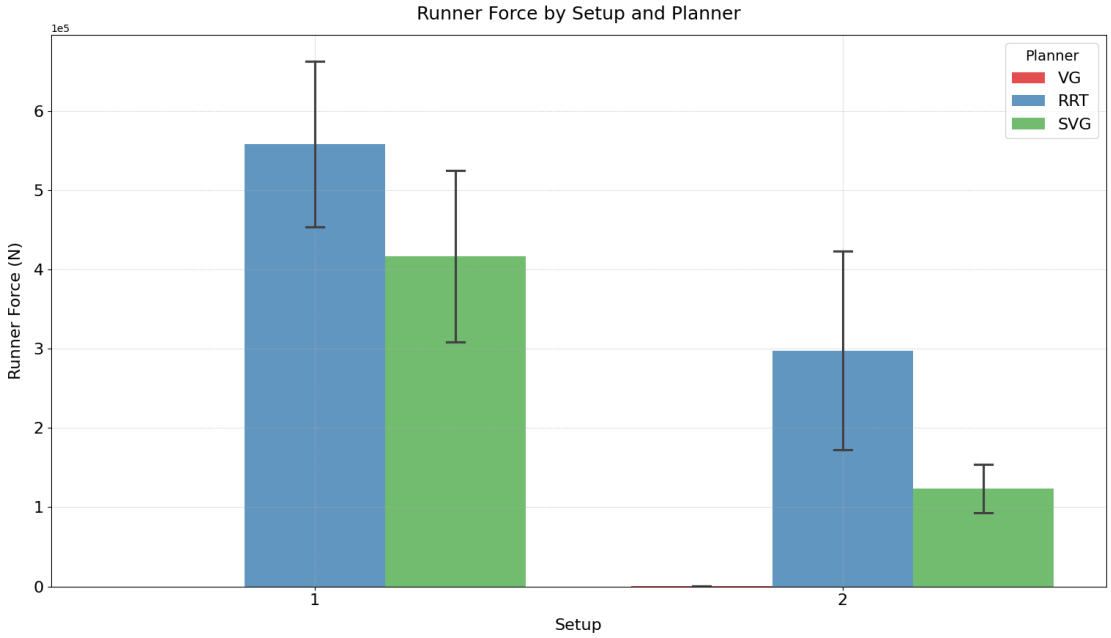


Figure 5.4: Cumulative push force for each planner over the entire trajectory.

6

Conclusions and Future Research

In this section, we summarize the key findings and results of our research and reflect on the initial motivations behind the study in our conclusion. We then offer recommendations for future research, highlighting the limitations of our solution and considering insights from other works addressing the same problem.

Conclusion

This research introduces SVG-MPPI, a novel solution for Navigation Among Movable Obstacles (NAMO) that emphasizes continuous movability, representing a significant advancement over traditional methods. Unlike conventional NAMO algorithms that rely on binary movability properties and require separate planners for different functions [17, 18, 23, 25–29], SVG-MPPI integrates continuous movability into a unified approach. This approach simplifies navigation and is not restricted to specific types of obstacle configurations (e.g., LP_1 or LP_2), effectively handling an arbitrary number of blocking obstacles.

SVG-MPPI is built on two key innovations: first, the Semantic Visibility Graph (SVG), an adaptation of the classical Visibility Graph (VG) that incorporates semantic information to determine the continuous movability of obstacles for path planning; second, the integration of this continuous movability into the local control strategy using Model Predictive Path Integral (MPPI). In SVG-MPPI, the mass of obstacles is used as a primary indicator of movability, represented within a distribution specific to each object category. Objects with mass below a threshold are considered movable, with lighter objects being more favorable for pushing. The MPPI strategy uses the IsaacGym physics engine and includes contact force minimization in its objective function to reduce redundant push actions and ensure interactions with obstacles are only as necessary to follow the path. This unified approach effectively merges global path planning with local control, addressing scenarios where the goal pose can be obstructed by an arbitrary number of blocking obstacles.

To the authors' knowledge, SVG-MPPI is the first approach to incorporate a continuous movability scale into planning strategies. Consequently, there are no existing comparative solutions or statistics in the literature. To benchmark SVG-MPPI, we compared it to two alternative global planning strategies: the basic Visibility Graph (VG) with no movability information and the Rapidly-exploring Random Tree (RRT), a sampling-based method with binary knowledge of obstacle movability. Each method was combined with MPPI as the local control strategy. The MPPI control received the same movability information as the global planner: VG-MPPI uses no movability information, RRT-MPPI uses binary movability information, and SVG-MPPI employs a continuous movability scale.

The solution was demonstrated through both qualitative and quantitative experiments. Qualitative experiments highlighted SVG-MPPI's effectiveness in navigating environments that required multiple push actions. The robot successfully created passages through obstacles, and in cases where initial estimations were incorrect, it was able to replan effectively. Real-world testing, detailed in Section 4.2, further confirmed SVG-MPPI's adaptability. The robot selected the most manageable obstacles to push and chose the path with the least effort, as shown in the results discussed in Section 5.1.

For the quantitative evaluation, described in Section 4.3, two different room setups were used to benchmark SVG-MPPI against other path planners: one with a fixed cluttered layout and another with randomized obstacles. The results from these quantitative experiments, presented in Section 5.2, demonstrate that SVG-MPPI outperforms both VG and RRT in path planning and execution success rates in the first setup. SVG's strategic placement of nodes allows for more effective navigation around obstacles, leading to higher success rates. Its approach ensures that paths of least resistance are chosen, making it particularly effective in environments requiring multiple push actions.

In the second setup with randomized obstacles, SVG again demonstrated superior performance compared to RRT, achieving higher success rates for both path planning and execution, as also shown in Section 5.2. Notably, SVG exhibited lower cumulative trajectory contact force, indicating fewer push actions and greater efficiency compared to RRT. Although RRT performs well in less structured environments, it is slower in waypoint calculation and less efficient in execution compared to SVG. VG, while offering the fastest planning times due to its simple obstacle avoidance strategy, fails in scenarios requiring obstacle pushing. In contrast, SVG achieves a reasonable deterministic planning speed close to real-time for the experiments, with effective navigation and reduced force application. Overall, SVG-MPPI proves to be a more reliable and efficient solution for Navigation Among Movable Obstacles (NAMO), combining high success rates with effective handling of complex scenarios.

In conclusion, SVG-MPPI has proven to be effective in navigating environments with movable obstacles by focusing on the concept of continuous movability. By integrating this continuous movability into both local and global planning strategies, SVG-MPPI enables the robot to dynamically push obstacles when necessary and maneuver around them when possible. The elimination of separate task, transit, and transfer planners provides a cohesive solution that combines strategic node placement with efficient path planning. Additionally, the system's capability to handle inaccurate movability estimations and adjust plans based on real-time evaluations highlights its robustness. Overall, SVG-MPPI represents a novel and modern approach to navigating complex environments with movable obstacles, demonstrating significant potential for further extension and application of the algorithm.

Future Research

Although SVG-MPPI has demonstrated promising results, several areas require further refinement to enhance its performance and practical application. Addressing these issues could significantly improve SVG-MPPI's reliability and effectiveness in navigating environments with movable obstacles.

One key area for improvement is the current approach to continuous movability, which is based solely on the mass of the obstacle. While mass is an important factor, movability also depends on additional physical properties such as friction and stability. Expanding the definition of movability to include these factors would enhance the algorithm's accuracy and effectiveness in diverse scenarios. The framework does allow for adjustments in movability estimation by modifying the node cost, but more comprehensive integration could be beneficial. Recent research into learning-based models, such as Large Language Models (LLMs) and reinforcement learning approaches, has shown promise in estimating these physical properties [70–72]. Additionally, analytical approaches for deriving object properties have been extensively studied and proven useful to derive physical properties of objects in the environment [73, 74].

Another area for enhancement is the simulation environment, which currently restricts obstacle representation to simple rectangular shapes. This simplification facilitates implementation but does not fully capture the complexity of real-world obstacles. To address this limitation, SVG-MPPI should consider transitioning to more advanced simulation and physics engines that can model a wider range of objects, such as chairs and tables. Research indicates that alternatives like IsaacGym, Omniverse, Mujoco, and DART offer greater flexibility and accuracy in simulating complex environments [75, 76].

To advance the SVG algorithm, addressing inefficiencies in distance calculations between geometric shapes can also be beneficial. The current brute-force method, with a quadratic complexity of $O(m \cdot n)$, becomes inefficient as the number of vertices increases [30]. Adopting more advanced methods, such as the Rotating Calipers approach [77], which offers $O(m + n)$ complexity, or the Gilbert-Johnson-Keerthi (GJK) Distance Algorithm [78, 79], could significantly improve efficiency. Integrating these techniques would improve performance in larger and more complicated scenarios.

Lastly, implementing gain scheduling could optimize the objective function for obstacles with varying movability. The current system faces a challenge: it aims to minimize push actions while allowing the robot to push obstacles when necessary. the objective function tends to behave differently for obstacles that are easily moved compared to those that are more difficult to move. Gain scheduling could address this by dynamically adjusting control parameters based on obstacle movability, enabling more precise tuning [80]. This would improve the algorithm's ability to balance minimizing push actions with effectively navigating around obstacles. Our current solution could benefit significantly from these improvements, as gain scheduling would enable more nuanced and adaptive control strategies, leading to better performance and efficiency in complex environments with diverse obstacles.

References

[1] Amir Aly, Sascha S Griffiths, and Francesca Stramandinoli. “Metrics and benchmarks in human-robot interaction: Recent advances in cognitive robotics”. In: *Cognitive Systems Research* 43 (2017), pp. 313–323.

[2] Arpita Soni. “Advancing Household Robotics: Deep Interactive Reinforcement Learning for Efficient Training and Enhanced Performance”. In: *Journal of Electrical Systems* 20.3s (Apr. 2024), pp. 1349–1355.

[3] Marina Paolanti et al. “Mobile robot for retail surveying and inventory using visual and textual analysis of monocular pictures based on deep learning”. In: *2017 European Conference on Mobile Robots (ECMR)*. 2017, pp. 1–6.

[4] Aarni Tuomi, Iis P Tussyadiah, and Jason L Stienmetz. “Applications and Implications of Service Robots in Hospitality”. In: *Cornell Hospitality Quarterly* 62 (2020), pp. 232–247.

[5] Omar Mubin et al. “A review of the applicability of robots in education”. In: *Technology for Education and Learning* 1 (Aug. 2013).

[6] Maria Kyrarini et al. “A Survey of Robots in Healthcare”. In: *Technologies* (2021).

[7] Subhodeep Mukherjee et al. “Humanoid robot in healthcare: A Systematic Review and Future Research Directions”. In: *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)*. Vol. 1. 2022, pp. 822–826.

[8] Juan Jesús Roldán et al. “Robots in agriculture: State of art and practical experiences”. In: *Service robots* 12.2 (2018), pp. 67–90.

[9] Andrea Botta et al. “A Review of Robots, Perception, and Tasks in Precision Agriculture”. In: *Applied Mechanics* 3.3 (2022), pp. 830–854.

[10] Srijeet Halder and Kereshmeh Afsari. “Robots in Inspection and Monitoring of Buildings and Infrastructure: A Systematic Review”. In: *Applied Sciences* 13.4 (2023).

[11] Theodoros Theodoridis and Huosheng Hu. “Toward intelligent security robots: A survey”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1219–1230.

[12] Deepak Patil et al. “A Survey On Autonomous Military Service Robot”. In: *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. 2020, pp. 1–7.

[13] 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.

[14] In Lee. “Service Robots: A Systematic Literature Review”. In: *Electronics* 10 (May 2021), p. 2658.

[15] Argentina Ortega, Nico Hochgeschwender, and Thorsten Berger. “Testing Service Robots in the Field: An Experience Report”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 165–172.

[16] Mike Stilman et al. “Planning and Executing Navigation Among Movable Obstacles”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 820–826.

[17] 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.

[18] 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.

[19] Thomas Collins et al. “Occupancy grid mapping: An empirical evaluation”. In: *2007 Mediterranean Conference on Control & Automation* (2007), pp. 1–6.

[20] Xiaoning Han et al. “Semantic Mapping for Mobile Robots in Indoor Scenes: A Survey”. In: *Information* 2 (2021).

[21] 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.

[22] 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).
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] 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.
[30] Mark De Berg. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000.
[31] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.
[32] 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.
[33] Han-Pang Huang and Shu-Yun Chung. "Dynamic visibility graph for path planning". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. 2004, pp. 2813–2818.
[34] Der-Tsai Lee. *Proximity and reachability in the plane.* University of Illinois at Urbana-Champaign, 1978.
[35] Subir Kumar Ghosh and David M Mount. "An Output-Sensitive Algorithm for Computing Visibility Graphs". In: *SIAM J. Comput.* 20 (1991), pp. 888–910.
[36] T K Priya and Krishnamurthy Sridharan. "A parallel algorithm, architecture and FPGA realization for high speed determination of the complete visibility graph for convex objects". In: *Microprocess. Microsystems* 30 (2006), pp. 1–14.
[37] Anurag Ganguli, Jorge Cortés, and Francesco Bullo. "Multirobot Rendezvous With Visibility Sensors in Nonconvex Environments". In: *IEEE Transactions on Robotics* 25 (2006), pp. 340–352.
[38] Sarah Masud et al. "Maximum visibility queries in spatial databases". In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)* (2013), pp. 637–648.
[39] 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.
[40] Corrado Pezzato et al. "Sampling-Based MPC Using a GPU-parallelizable Physics Simulator as Dynamic Model: an Open Source Implementation with IsaacGym". In: 2023.
[41] Mohak Bhardwaj et al. *STORM: An Integrated Framework for Fast Joint-Space Model-Predictive Control for Reactive Manipulation*. 2021.
[42] James Jerry Gibson. *The theory of affordances*. Psychology Review, 1977.
[43] Paola Ard'on et al. "Affordances in Robotic Tasks - A Survey". In: *ArXiv* abs/2004.07400 (2020).
[44] Huaqing Min et al. "Affordance Research in Developmental Robotics: A Survey". In: *IEEE Transactions on Cognitive and Developmental Systems* 8 (2016), pp. 237–255.

[45] Timo Lüddecke, Tomas Kulvicius, and Florentin Wörgötter. “Context-based affordance segmentation from 2D images for robot actions”. In: *Robotics Auton. Syst.* 119 (2019), pp. 92–107.
[46] Joanna McGrenere and Wayne Ho. “Affordances: Clarifying and Evolving a Concep”. In: *Graphics Interface*. 2000.
[47] Maozhen Wang et al. “Affordance-Based Mobile Robot Navigation Among Movable Obstacles”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 2734–2740.
[48] Peter Kaiser et al. “Validation of whole-body loco-manipulation affordances for pushability and liftability”. In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 2015, pp. 920–927.
[49] Clara Gomez et al. “Object-Based Pose Graph for Dynamic Indoor Environments”. In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5401–5408.
[50] Thanh-Toan Do et al. “AffordanceNet: An End-to-End Deep Learning Approach for Object Affordance Detection”. In: *CoRR* abs/1709.07326 (2017).
[51] Nicholas Watters et al. “Visual Interaction Networks”. In: *CoRR* abs/1706.01433 (2017).
[52] Steven M Hyland, Jing Xiao, and Cagdas D Onal. “Predicting Center of Mass by Iterative Pushing for Object Transportation and Manipulation”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2023, pp. 1615–1620.
[53] Mathew Halm and Michael Posa. “A Quasi-static Model and Simulation Approach for Pushing, Grasping, and Jamming”. In: *CoRR* abs/1902.03487 (2019).
[54] Tsuneo Yoshikawa and Masamitsu Kurisu. “Indentification of the center of friction from pushing an object by a mobile robot”. In: *Proceedings IEEE/RSJ International Workshop on Intelligent Robots and Systems 1991* (1991), pp. 449–454.
[55] Michael B Chang et al. “A Compositional Object-Based Approach to Learning Physical Dynamics”. In: *CoRR* abs/1612.00341 (2016).
[56] Julian Busch, Jiaxing Pi, and Thomas Seidl. “PushNet: Efficient and Adaptive Neural Message Passing”. In: *CoRR* abs/2003.02228 (2020).
[57] Jiacheng Yuan et al. “Active Mass Distribution Estimation from Tactile Feedback”. In: *ArXiv* (2023).
[58] 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.
[59] Trevor Scott Standley et al. “image2mass: Estimating the Mass of an Object from Its Image”. In: *Conference on Robot Learning*. 2017.
[60] Jiajun Wu et al. “Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning”. In: *Neural Information Processing Systems*. 2015.
[61] 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.
[62] Dimitri Bertsekas. *Convex optimization theory*. Vol. 1. Athena Scientific, 2009.
[63] Saman Ahmadi et al. “Bi-objective Search with Bi-directional A”. In: *CoRR* abs/2105.11888 (2021).
[64] Maxim Likhachev et al. “Anytime Dynamic A*: An Anytime, Replanning Algorithm.” In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. Aug. 2005, pp. 262–271.
[65] Sven Koenig and Maxim Likhachev. “Real-time adaptive A*”. In: *Adaptive Agents and Multi-Agent Systems*. 2006.
[66] Anthony Stentz. “The focussed D* algorithm for real-time replanning”. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI’95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 1652–1659. ISBN: 1558603638.
[67] Agarwal Ravi P and Patricia J Y Wong. *Spline Interpolation*. Dordrecht: Springer Netherlands, 1993, pp. 281–362.
[68] Jovina Seau Ling Leong, Kenneth Tze Kin Teo, and Hou Pin Yoong. “Four Wheeled Mobile Robots: A Review”. In: *2022 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*. 2022, pp. 1–6.
[69] Mark Payton, Matthew Greenstone, and Nathaniel Schenker. “Overlapping Confidence Intervals or Standard Error Intervals: What Do They Mean in Terms of Statistical Significance?” In: *Journal of insect science (Online)* 3 (Aug. 2003), p. 34.

[70] Marek Kopicki et al. “One-shot learning and generation of dexterous grasps for novel objects”. In: *The International Journal of Robotics Research* 35 (2016), pp. 959–976.
[71] Jochen Stüber, Marek Sewer Kopicki, and Claudio Zito. “Feature-Based Transfer Learning for Robotic Push Manipulation”. In: *CoRR* abs/1905.03720 (2019).
[72] Haoran Geng et al. *SAGE: Bridging Semantic and Actionable Parts for GEneralizable Manipulation of Articulated Objects*. 2024.
[73] Jochen Stüber, Claudio Zito, and Rustam Stolkin. “Let’s Push Things Forward: A Survey on Robot Pushing”. In: *CoRR* abs/1905.05138 (2019).
[74] Andrej Kruzliak et al. *Interactive Learning of Physical Object Properties Through Robot Manipulation and Database of Object Measurements*. 2024.
[75] Youngsung Son, Hyonyong Han, and Joonmyun Cho. “Usefulness of using Nvidia IsaacSim and IsaacGym for AI robot manipulation training”. In: *2023 14th International Conference on Information and Communication Technology Convergence (ICTC)* (2023), pp. 1725–1728.
[76] Michael Kaup et al. *A Review of Nine Physics Engines for Reinforcement Learning Research*. 2024.
[77] Godfried T Toussaint. “The Rotating Calipers: An Efficient, Multipurpose, Computational Tool”. In: The International Conference on Computing Technology and Information Management (ICCTIM), 2016.
[78] Chong Jin Ong and E G Gilbert. “The Gilbert-Johnson-Keerthi distance algorithm: a fast version for incremental motions”. In: *Proceedings of International Conference on Robotics and Automation*. Vol. 2. 1997, pp. 1183–1189.
[79] Yu Zheng and Katsu Yamane. “Generalized Distance Between Compact Convex Sets: Algorithms and Applications”. In: *IEEE Transactions on Robotics* 31 (2015), pp. 988–1003.
[80] Wilson J Rugh and Jeff S Shamma. “Research on gain scheduling”. In: *Automatica* 36.10 (2000), pp. 1401–1425.

Appendices

A. Nomenclature

Abbreviations

Abbreviation	Definition
NAMO	Navigation Among Movable Obstacles
VG	Visibility Graph
SVG	Semantic Visibility Graph
RRT	Rapidly Random Exploring Random Tree
MPPI	Model Predictive Path Integral
URDF	Unified Robot Description Format

Symbols

Symbol	Definition
V	Set of all vertices in a graph
E	Set of edges connecting vertices in V
S	Set of obstacles in the environment
r	Safety margin used around obstacles
p	Passage node placed between obstacles
d	Distance metric between nodes or obstacles
l	Length of a boundary or path
t	Time or an interpolation factor
A, B	Sets or regions of interest
C	Combined or resulting area from operations on A and B
m	Mass or other scalar property of an object
T	Time horizon for planning or analysis
K	Number of iterations, samples, or elements in a set
$\mathbf{x}$	State vector representing a position or condition
$\mathbf{v}$	Control input vector or velocity
γ	Discount factor or scaling coefficient
$\mathbf{F}$	Force vector or matrix
$\mathbf{P}$	Set of waypoints or positions
$\mathbf{c}$	Cost function or vector
$\mathbf{W}$	Weight matrix for optimization or control
α	Normalization or scaling factor

B. Rapidly Exploring Random Tree (RRT)

The Rapidly Exploring Random Tree (RRT) algorithm is a popular method used in robotics and motion planning for efficiently searching non-convex, high-dimensional spaces. It incrementally builds a space-filling tree, which helps in exploring feasible paths from an initial position to a target goal while avoiding obstacles. The main idea of RRT is to sample points randomly in the search space and incrementally build a tree that extends towards these random points. This allows the tree to rapidly explore the space, hence the name.

As can be seen in Table 1.1, solutions to (a subset of) NAMO problems, also often use this algorithm to produce a path from a starting pose to a goal pose. Algorithm 7 presents how RRT calculates the path towards the goal. The modification in blue is implemented to provide a fair comparison to the modified SVG algorithm, explained in Chapter 3. Each step, including the modification, is explained afterwards.

Algorithm 4: Rapidly-exploring Random Trees (RRT)

```
1 V ← {x_init}; E ← ∅;
2 for i = 1, ..., n do
3     x_rand ← sampleFree_i;
4     x_nearest ← nearest(G = (V, E), x_rand);
5     if obstacleFree(x_nearest, x_new) then
6         V ← V ∪ {x_new}; E ← E ∪ {(x_nearest, x_new)};
7 return G = (V, E);
```

The Rapidly Exploring Random Tree (RRT) algorithm starts by initializing two sets: one for vertices (V) and one for edges (E). The vertex set begins with only the initial point x_{init}, while the edge set is empty initially. The algorithm then enters a loop that runs for a specified number of iterations (n). During each iteration:

1. A random point x_{rand} is selected within the free space.
2. The nearest existing vertex x_{nearest} in the tree to x_{rand} is identified.
3. A new point x_{new} is generated by moving from x_{nearest} towards x_{rand} by a fixed step size.
4. A collision check is performed to ensure the path from x_{nearest} to x_{new} is clear of obstacles, without obstacle inflation considered.
5. If the path is obstacle-free, the new point x_{new} is added to the set of vertices (V), and the edge connecting x_{nearest} to x_{new} is added to the set of edges (E).

Traditionally, collision check is performed on inflated obstacles, which represent the configuration space, to ensure a safe distance from all the obstacle. However, because of the comparison to SVG samples are also allowed to be close to the border as the modified algorithm also allows passage nodes to be as close to obstacle borders. This modification is within the *obstacleFree* method, colored blue in the Algorithm 7.

After completing the loop, the algorithm returns the tree structure ($G = (V, E)$), representing the explored space. The RRT algorithm is particularly valuable in navigating complex and high-dimensional environments where traditional grid-based methods are impractical. Its ability to quickly explore the search space in a random manner makes it highly effective in robot motion planning and related applications.

C. Hardware Setup

In this section, we provide the detailed specifications for the experimental framework used in our study. Table 3 summarizes the key parameters of the hardware and simulation setup. This table includes information on the robot dimensions, weight, maximum push mass, and various settings for the path planning and control algorithms employed during the experiments. The parameters listed are important for understanding the operational limits and configurations under which the experiments were conducted.

Table 3: Experimental Framework Parameters Specifications

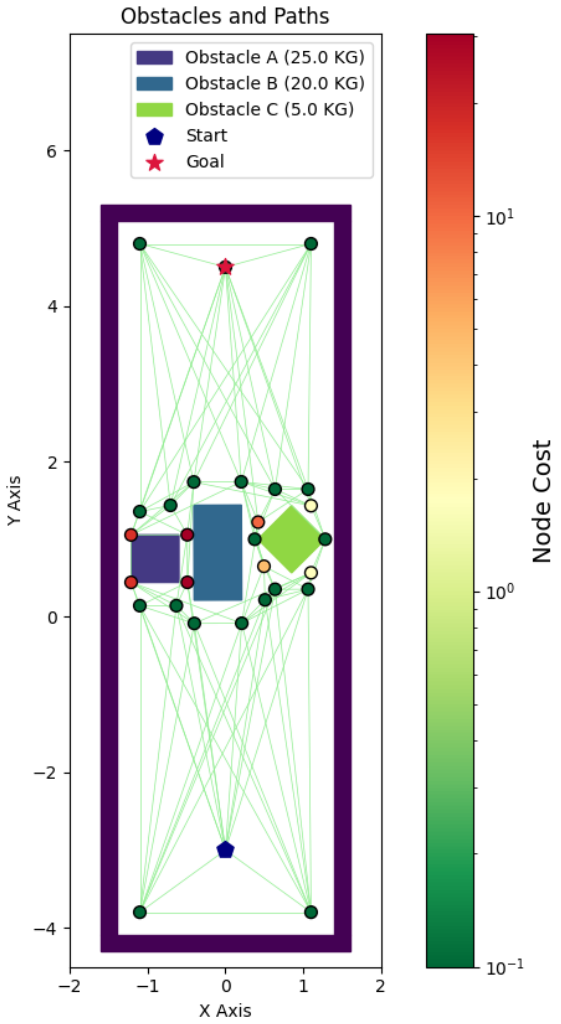
Parameter	**Value**	**Unit**
Robot Dimensions	686 x 517 x 114	mm
Robot Weight	9.1	kg
Maximum Push Mass	30	kg
Safety Inflation	0.3	meters
Mass Threshold	30	kg
Path Inflation	0.3	meters
Spline Interval	0.5	meters
Replan Timing	30	seconds
Simulation Time Step (dt)	0.08	seconds
MPPI Mode	”halton-spline”	-
Sampling Method	”halton”	-
Number of Samples	1000	-
Horizon	25	-
Lambda	0.05	-
Control Input Min	[-0.2, -0.2, -0.4]	m/s, m/s, rad/s
Control Input Max	[0.4, 0.4, 0.4]	m/s, m/s, rad/s
Noise Sigma	$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$	-
Update Covariance	True	-
Rollout Variance Discount	1.0	-
Filter Control Input	True	-

D. Experiment Graphs and Paths

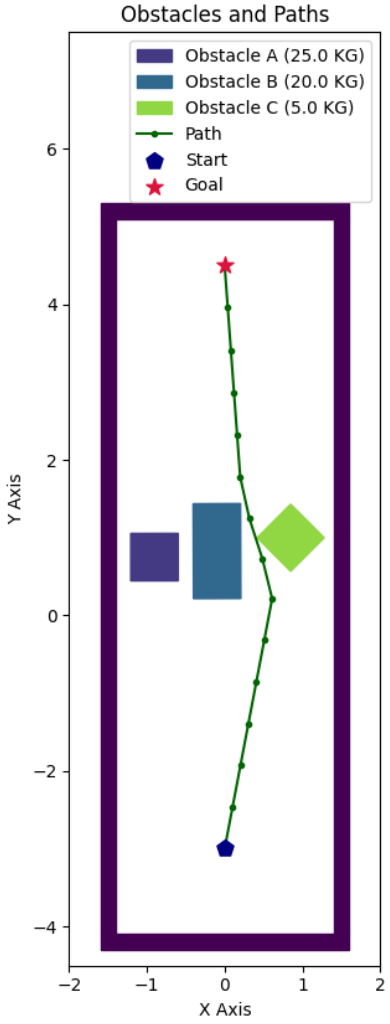
This section provides additional visual insights into the robot's navigation performance in both real-world and simulated experiments. The following figures illustrate the robot's interaction with obstacles and its path-planning strategies, helping to understand how the robot navigates different environments.

Real-World Qualitative Experiment

The real-world qualitative experiment demonstrates the robot's ability to navigate around obstacles and select paths in a practical setting. The figures below show the weighted graph of obstacles and the path calculated by the robot during the real-world experiment.



(a) Graph showing the obstacles and possible edges in the real-world environment. The layout includes obstacle positions and the path selected by the robot.

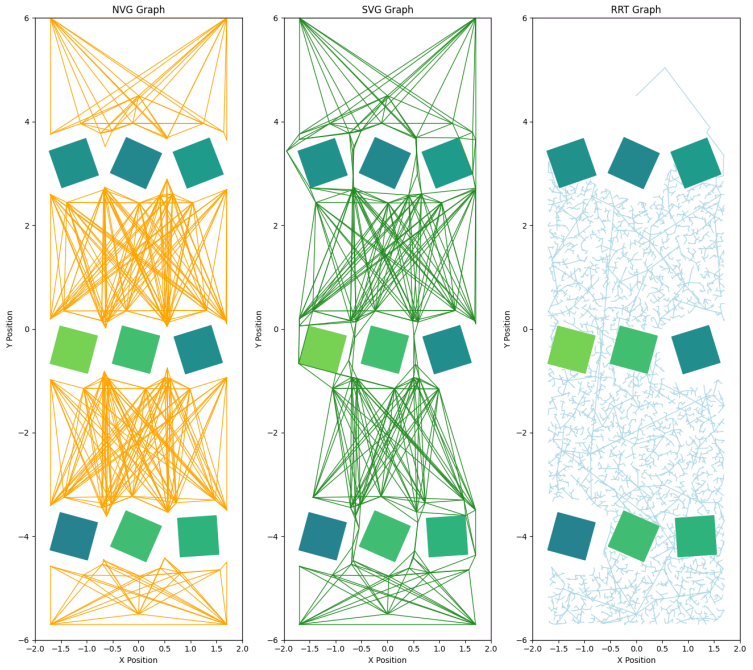


(b) Path visualization of the robot in the real-world scenario. It highlights the shortest path calculated by the algorithm.

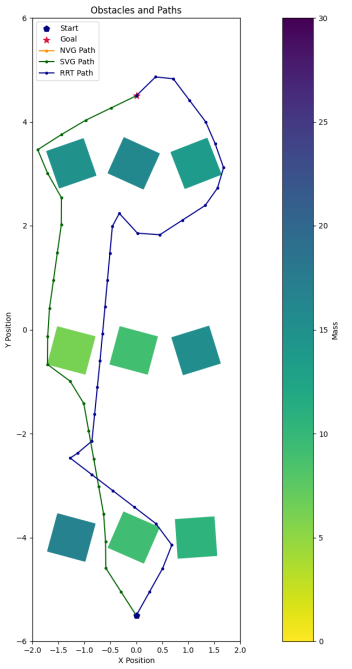
Figure 1: Visualizations of the robot's navigation in the real-world experiment: (a) Weighted graph of obstacles and edges; (b) The calculated path taken by the robot.

Quantitative Experiments

The quantitative experiments provide a comparative analysis of different planners (VG, RRT, and SVG) in two different setups. The figures below illustrate the weighted graphs and shortest paths computed by each planner in an example of Setup 1 and Setup 2.

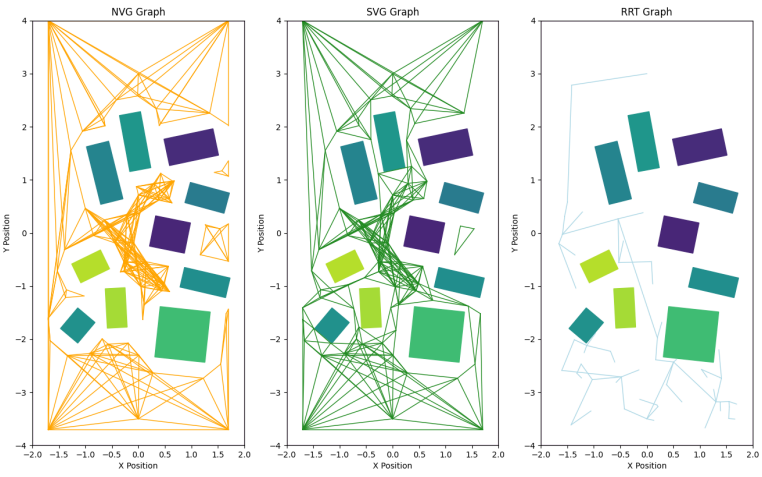


(a) Weighted graphs for different planners (VG, RRT, and SVG) in Setup 1. The graph displays obstacles and the edges considered by each planner to determine the shortest path.

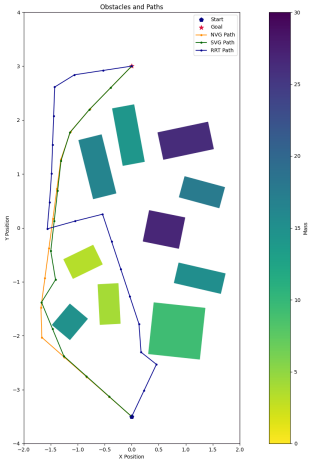


(b) Shortest paths computed by different planners in Setup 1.

Figure 2: Visualizations for Setup 1: (a) Weighted graphs showing obstacles and edges for each planner; (b) Shortest paths computed by the planners.



(a) Weighted graphs for different planners (VG, RRT, and SVG) in Setup 2. The graph displays obstacles and the edges considered by each planner to determine the shortest path in a new obstacle configuration.



(b) Shortest paths computed by different planners in Setup 2.

Figure 3: Visualizations for Setup 2: (a) Weighted graphs showing obstacles and edges for each planner; (b) Shortest paths computed by the planners.