2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Interactive-FAR:Interactive, Fast and Adaptable Routing for Navigation Among Movable Obstacles in Complex Unknown Environments

Botao He † [1], Guofei Chen † [2], Wenshan Wang [2], Ji Zhang* [2], Cornelia Fermuller[1], and Yiannis Aloimonos [1]

***Abstract*— This paper introduces a real-time algorithm for navigating complex unknown environments cluttered with movable obstacles. Our algorithm achieves fast, adaptable routing by actively attempting to manipulate obstacles during path planning and adjusting the global plan from sensor feedback. The main contributions include an improved dynamic Directed Visibility Graph (DV-graph) for rapid global path searching, a real-time interaction planning method that adapts online from new sensory perceptions, and a comprehensive framework designed for interactive navigation in complex unknown or partially known environments. Our algorithm is capable of replanning the global path in several milliseconds. It can also attempt to move obstacles, update their affordances, and adapt strategies accordingly. Extensive experiments validate that our algorithm reduces the travel time by 33%, achieves up to 49% higher path efficiency, and runs faster than traditional methods by orders of magnitude in complex environments. It has been demonstrated to be the most efficient solution in terms of speed and efficiency for interactive navigation in environments of such complexity. We also open-source our code in the docker demo[1] to facilitate future research.**

I. Introduction

Interactive navigation is a navigation task that involves interactions between the robot and movable obstacles. Environmental interaction is essential especially for robots navigating cluttered spaces. This paper proposes a solution for interactive navigation in cluttered unknown environments, focusing on fast and adaptable navigation with environmental interactions. The problem remains challenging as it needs to consider multiple sub-tasks simultaneously, which include: 1) Dynamically updating the environment representation with new sensor data. 2) Executing a global path planning to avoid local minima, which considers environmental interactions during the path search. 3) Strategizing interaction planning to manipulate objects. In cases where the environment is complex with numerous movable obstacles, the problem's computational complexity increases significantly. Consequently, ensuring the algorithm operates in real time becomes a challenge.

Our work proposes a real-time framework that consists of identification / mapping, path planning, and interaction. This is achieved through two key ideas.

The first idea is a hybrid map representation called Directed Visibility Graph (DV-graph). This graph's edges represent roads and the traversability between the connected vertices. Some of the edges also encode strategies on how to manipulate movable obstacles away from the path. The sparse nature of DV-Graph and its integrated interaction strategies allow for a global path search with low latency (within 10ms on a 4.7GHz i7) while also considering environmental interactions. DV-graph is updated hierarchically at each data frame. It consists of two layers. The local layer converts the sensor data into a polygon map, on which interaction planning for each movable obstacle is performed. The prospective strategies are encoded into visibility edges and connected between polygons to form the DV-graph. The global layer is a larger DV-graph that is merged and

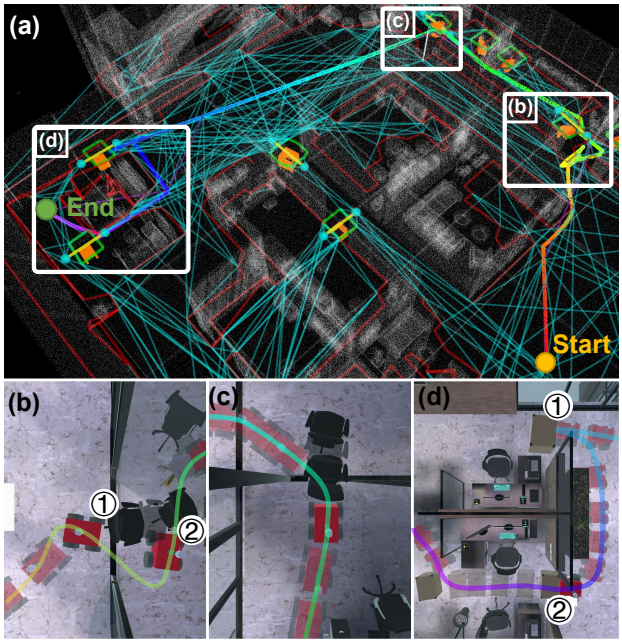

Fig. 1: Illustration of the interactive navigation through an unknown environment. The colorful curve is the vehicle's trajectory. The obstacles are extracted as polygons in red, and the movable objects (orange pixels) are registered as green polygons. The cyan dots represent topological waypoints. The Directed Visibility Graph is marked as cyan and yellow lines. (a) Overview. (b-c) The robot can manipulate the movable obstacles during the navigation and can switch the contact points during manipulation. (d) The robot attempts to move a heavy object ①. When the robot contacts the object, the cost of moving it is updated from force feedback. If the object is affordable, the robot pushes it, otherwise it replans the alternate strategy to execute. The robot can quickly adapt to new sensory observations no matter whether a movable object is affordable or not.

† **Equal contribution.**

1 Perception and Robotics Group, University of Maryland, MD 20742.
2 Robotics Institute, Carnegie Mellon University, PA 15213-3890.
Email: botao@umd.edu, {guofei,zhangji}@cmu.edu

[1]Interactive-FAR: github.com/Bottle101/Interactive-FAR

U.S. Government work not protected by U.S. copyright

updated with the local layer at each frame. The hierarchical and incremental graph construction framework ensures real-time performance during navigation. This framework also supports manipulation actions such as push, pull, and pick-up. These can be encoded into the same map representation as a multi-level DV-graph for rapid global path search.

The second key idea is adaptable interaction planning via kinodynamic path search. It makes the robot actively attempt to move obstacles, update their affordances, and adapt strategies accordingly. Given a local region divided into multiple components by a movable obstacle with unknown physics properties, as shown in Fig. 3, the planner tries to generate a series of kinodynamically feasible manipulation strategies that reconnect sub-regions based on assumed physical properties. These strategies enable the robot to clear the object from its path. During the navigation, the robot attempts to move the object by executing the above strategy, the planner updates the physics properties, thus updating the affordance, of the object through sensory feedback. The robot continues executing the original strategy if the estimated affordance is within a certain range. Otherwise, the whole global path is replanned. The updated affordance is then stored in the DV graph for later local and global planning.

The proposed algorithm is validated in complex environments of different scales and with movable obstacles. Extensive experiments show that our algorithm is faster than most benchmarks in the order of magnitudes while maintaining up to 49% higher path efficiency.

The main contributions can be summarized as:

- An improved dynamic DV-graph that encodes the interaction strategies and accelerates path finding.
- An interaction planning method that adapts to the online physics property evaluation of movable obstacles.
- A complete navigation algorithm for interactive navigation in unknown or partially known environments.

We open-source our code in the docker demo to facilitate future research.

II. Related Work

1) Navigation in Unknown Environments: Navigation in unknown environments has been studied and applied in many field applications, such as for exploration [1, 2] and swarm formation [3]. However, these works regard all obstacles as static and fixed, which limits the robot's ability to move in cluttered environments.

2) Interactive Navigation -Geometry Based: Previous studies have formulated this problem as a geometric motion planning problem called navigation among movable objects (NAMO). Wilflong [4] proved that NAMO is NP-hard. Stilman et al. [5] derived optimal solutions for 2D geometric planning for a subset of NAMO problems called $LP1$ with complete knowledge of the environment and objects. Levihn et al. [6] proposed a locally optimal algorithm for $LP1$ NAMO problems in partially known environments, and Muguira-Iturral et al. [7] takes visibility into account in NAMO problem. However, solving the NAMO problem on a grid map is time-consuming [5], the manipulation actions are restricted to fixed contact points or directions, and the observation of object shape is assumed to be perfect.

3) Interactive Navigation -Learning Based: Recent works that mapped visual input to agents' actions using reinforcement learning include [8] and [9]. However, the agent relies on a precomputed shortest path to operate in [8], which is hard to achieve in real-world settings, and the interaction is simplified to moving a single object with a few predefined actions to reach a goal within its sensor range in [9].

This work focuses on developing a real-time adaptable interactive navigation algorithm in complex environments where the approaches above fail. We introduce a DV-graph-based mapping and path-searching algorithm that facilitates rapid path planning in complex environments. The adaptable interaction planning algorithm empowers the robot to attempt obstacle manipulation, infer the manipulation affordance via tactile sensing, update the navigation graph, and adapt its strategies according to the latest navigation graph.

III. Problem Definition

Our problem is divided into two sub-problems: interactive motion planning and adaptive manipulation.

For the interactive motion planning problem, define $\mathcal{Q} \subset \mathbb{R}^3$ as the work space for the robot to navigate. Let $\mathcal{S} \subset \mathcal{Q}$ be the perceived sensor data. Here, $\mathcal{S}$ is classified as the object-level point clouds, which contains the points of movable objects $\mathcal{S}_{mov} \subset \mathcal{S}$ and points of the background obstacles $\mathcal{S}_{bg} = \mathcal{S} \backslash \mathcal{S}_{mov}$. We proposed a directed visibility graph (DV-graph) denoted as $\mathcal{G} = (V, \Pi)$, from $\mathcal{S}$, where $\mathbf{v_i} \in V$ represents one vertex and $\langle \mathbf{v_p}, \mathbf{v_q} \rangle \in \Pi$ means an ordered vertex pair in $\mathcal{G}$. The problem can be expressed as:

Problem 1: Given the robot position $\mathbf{p}_{robot} \in \mathcal{Q}$ and goal point $\mathbf{p}_{goal} \in \mathcal{Q}$, find the most energy-efficient path between $\mathbf{p}_{robot}$ and $\mathbf{p}_{goal}$ on $\mathcal{G}$. The energy efficiency considers both the travel distance and effort efficiency when manipulating objects to move them out of the way.

Problem 1 is solved repetitively in each planning cycle. During navigation, we online update $\mathcal{G}$ with new sensor data, mapping the newly perceived environment and updating the maintained global map. Then, we re-plan the path on the updated $\mathcal{G}$ in the next planning cycle until arriving at $\mathbf{p}_{goal}$.

For adaptive manipulation, define $\mathcal{Q}_{local} \subset \mathcal{Q}$ as the local workspace for a single manipulation. Let $\{\mathcal{C}^i_{local} \subset \mathcal{Q}_{local} | i \in \mathbb{Z}^+\}$ be locally disconnected components that are separated by a movable object, as shown in Fig. 3. The manipulation strategies are defined as motion primitives π. The problem is described as:

Problem 2: Given $\mathcal{Q}_{local}$ that is divided into n components $\{\mathcal{C}_{local}\}$ by a movable object, find $n(n-1)$ manipulation primitives π that re-connect any two components.

Problem 2 is solved once for each object that blocks a local space. The manipulation primitives are stored in the map with the object and utilized in two ways. First, it helps the DV-graph determine the travel cost from one component to another to trade off between manipulating or taking a

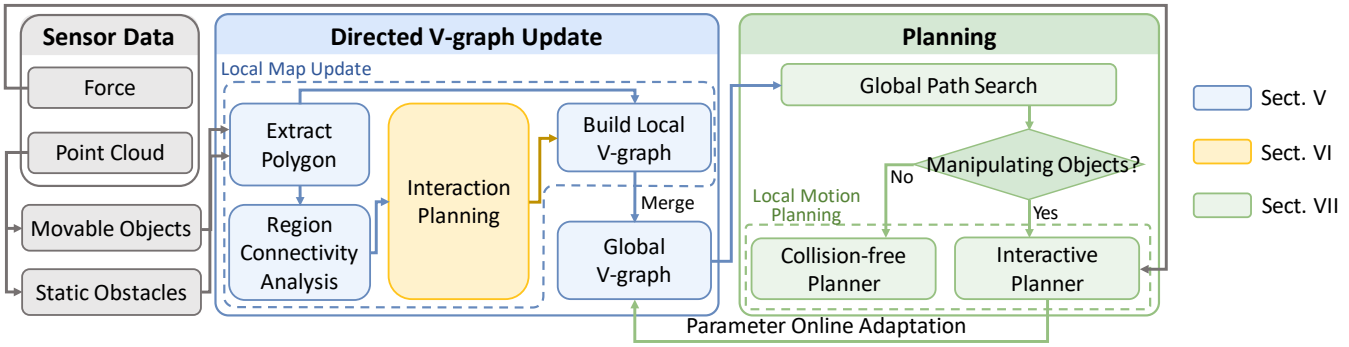

Fig. 2: A diagram of our system architecture. The algorithm comprises two primary components, DV-graph update is detailed in Sects. V and VI, planning algorithm is detailed in Sect. VII.

collision-free path. Second, the searched primitives guide the robot's actual manipulation motion controller. The reason we need $n(n-1)$ primitives is explained in Sect. VI.A.

In this paper, our assumption follows the definition of $LP1$, that one manipulation primitive only moves one single object, and that will not cause a new collision. We also assume that each Q_{local} is blocked by one object, otherwise it creates a new Q_{local}.

IV. System Overview

The architecture of the proposed algorithm is shown in Fig. 2. The algorithm comprises two primary components: DV-graph construction and updating (Sects. V and VI), and motion planning (Sect. VII). In the DV-graph construction and update phase, the process begins with pre-processed sensor data $\mathcal{S}_{mov}$ and $\mathcal{S}_{bg}$. Then, a polygon extraction procedure transforms $\mathcal{S}_{mov}$ and $\mathcal{S}_{bg}$ into two sets of polygons. These sets form the map representation for the DV-graph.

The topological space connectivity analysis algorithm is designed to determine whether a single movable object divides a local space into several disconnected components. The interaction planning module is employed for each movable object. This module generates manipulation strategies to reconnect these locally disconnected spaces and calculates the associated manipulation costs. The output of the simulation module is a comprehensive set of manipulation strategies, encompassing all feasible scenarios considering both the starting and ending states. Finally, the local DV-graph is constructed with the extracted polygons and the manipulation strategies in order to encode both the geometry and interaction information. The local DV-graph is merged with the global V-graph in each running cycle, facilitating its dynamic update.

In the planning phase, the process initially searches a global path from $\mathbf{p}_{robot}$ to $\mathbf{p}_{goal}$. During the path execution , the choice of local planner depends on the robot's its current navigation context: a collision-free planner is employed when the robot navigates between two visible waypoints. An interactive planner is utilized for mobile manipulation if the current local path segment intersects with movable objects, an interactive planner is utilized for mobile manipulation. During the manipulation, affordance of the object are dynamically updated. The interactive planner then adaptively revises the manipulation policy based on these updated parameters. Subsequently, these revised parameters are incorporated to update the corresponding edges in the global DV-graph.

V. DV-graph Construction and Update

A. Polygon Extraction

To differentiate between static and movable objects in the DV-graph, this module takes in $\mathcal{S}_{mov}$ and $\mathcal{S}_{bg}$ separately and transform them into two sets of polygons, denoted as $\{\mathcal{P}^i_{mov} \subset \mathcal{Q} | i \in \mathbb{Z}^+\}$ and $\{\mathcal{P}^i_{bg} \subset \mathcal{Q} | i \in \mathbb{Z}^+\}$. To achieve that, we first inflate $\mathcal{S}_{mov}$ and $\mathcal{S}_{bg}$ based on the robot dimension, and register them to 2-D image planes. Utilizing the methods in [10] and [11], enclosed polygons are extracted and simplified. This polygon extraction step is developed from [12], detailed explanation is available therein. Finally, the two sets of polygons are combined into a complete local polygon graph, denoted as $\{\mathcal{P}^i_{local} \subset \mathcal{Q} | i \in \mathbb{Z}^+\}$, as shown in Fig. 1(a). The $\mathcal{P}_{bg}$ are marked as red polygons and $\mathcal{P}_{mov}$ are marked as green. Notably, polygons from different classes can intersect, but no intersections occur within the same class due to the topological structure analysis based on [10].

B. Topological Space Connectivity Analysis

The free-space of the static background polygon map, denote as $\mathcal{Q} \backslash \{\mathcal{P}^i_{bg}\}$, is guaranteed to be a path-connected space [13] because of the topological structure analysis. This implies that for any point pairs $\langle \mathbf{p}_i, \mathbf{p}_j \rangle \in \mathcal{Q} \backslash \{\mathcal{P}^i_{bg}\}$, there always exist a connected path. However, when $\{\mathcal{P}^i_{bg}\}$ and $\{\mathcal{P}^i_{mov}\}$ are combined, certain regions become locally path-disconnected, as shown in Fig. 3. The space is divided into several locally disconnected components, $\{\mathcal{C}^i_{local} \subset \mathcal{Q} | i \in \mathbb{Z}^+\}$. To navigate through these disconnected spaces, the robot needs to actively manipulate the movable object to re-establish connectivity between regions. As illustrated in Fig. 3(a)-(b), the number of disconnected components leads to $n(n-1)$ potential manipulation strategies based on the robot's start and goal regions.

As shown in Fig. 3(c), For each $\mathcal{C}^i_{local}$, define $\mathbf{p}^i_{topo} \in \mathcal{C}^i_{local}$ as the representative point of $\mathcal{C}^i_{local}$, which is called topological waypoints because different $\mathbf{p}^i_{local}$ belongs to different topological spaces. To make topological waypoints

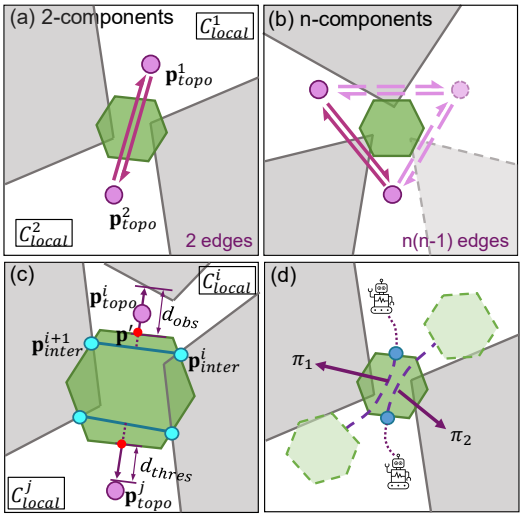

Fig. 3: (a-b) Path-disconnected components $\{\mathcal{C}^i_{local}\}$ of a local region $\{\mathcal{P}^i_{local}\}$. Green polygon represents $\mathcal{P}_{mov}$ and grey ones are $\mathcal{P}_{bg}$. They are inflated based on the robot dimension. Topological waypoints $\{\mathbf{p}_{topo}\}$ are marked as purple, potential manipulation strategies are represented as directed edges in purple. (c) Demonstration of the process to choose $\{\mathbf{p}^i_{topo}\}$. Intersected points $\mathbf{p}_{inter}$ are marked in cyan. (d) Illustration of the interaction planning module. Contact point are marked as blue.

have good visibility with other vertices in $\mathcal{C}^i_{local}$, the following strategy is adopted. Given a pair of consecutive intersection points $\langle \mathbf{p}^i_{inter}, \mathbf{p}^{i+1}_{inter} \rangle$, marked as cyan in Fig. 3(c), calculate its mid-point $\mathbf{p}_{mid}$ and normal vector $\vec{\mathbf{n}} = \overrightarrow{\mathbf{p}^i_{inter}\mathbf{p}^{i+1}_{inter}} \times (0,0,1)$. Then create its normal vector starts from $\mathbf{p}_{mid}$, and calculate its intersection point with the polygon, denote as $\mathbf{p}'$ and mark as red point in the figure. Finally, $\mathbf{p}^i_{topo}$ can be calculated as:

$$\mathbf{p}^i_{topo} = \mathbf{p}' + \vec{\mathbf{n}} \cdot \min(d_{thres}, \frac{d_{obs}}{2}), \tag{1}$$

where d_{obs} represents the minimal distance from $\mathbf{p}'$ to its closest obstacle along $\vec{\mathbf{n}}$, d_{thres} is a pre-defined threshold value with unit meters, here we choose 0.5 as the d_{thres}.

Theorem 1: There are at least three vertices in $\mathcal{C}^i_{local}$ visible (connected by a straight line in free space) from $\mathbf{p}^i_{topo}$.

Proof: To prove it, two axioms are introduced: i) Any polygon can be decomposed into triangles. ii) Any point in a triangle is visible to all three vertices. Given $\mathcal{C}^i_{local}$, it can be decomposed to a set of triangles $\{\Delta_k \subset \mathcal{C}^i_{local} | k \in \mathbb{Z}^+\}$, because $\mathbf{p}^i_{topo}$ is within $\mathcal{C}^i_{local}$, it must be in one of $\{\Delta_k\}$, thus connecting at least three vertices in $\mathcal{C}^i_{local}$.

If the robot can manipulate the object from $\mathcal{C}^i_{local}$, and achieves $\mathcal{C}^j_{local}$, then the point pair $\langle \mathbf{p}^i_{topo}, \mathbf{p}^j_{topo} \rangle$ is connected added into the DV-graph as a travelsable edge. The travelsibility of the edge is assigned based on the estimated cost of the manipulation.

C. Interaction Planning

Given a locally path-disconnected region with n-components, as shown in Fig. 3(a)-(b), the module is designed to generate $n(n-1)$ local manipulation policies

Algorithm 1: DV-graph Update

Input : Segmented Sensor Data: $\mathcal{S}_{bg}, \mathcal{S}_{mov}$, DV-graph: $\mathcal{G}$
Output: Updated DV-graph: $\mathcal{G}$

1 $\{\mathcal{P}_{bg}, \mathcal{P}_{mov}\} \leftarrow$ PolygonExtraction($\mathcal{S}_{bg}, \mathcal{S}_{mov}$);
2 ▷ Local DV-graph update
3 Construct local DV-graph on $\{\mathcal{P}^i_{bg}\}$;
4 **for** *each* $\{\mathcal{P}_{mov}\}$ **do**
5 **for** *each pair* $\langle \mathbf{p}^i_{topo}, \mathbf{p}^j_{topo} \rangle$ **do**
6 $\mathcal{J}, \pi \leftarrow \Gamma(\langle \mathbf{p}^i_{topo}, \mathbf{p}^j_{topo} \rangle, \mu)$
7 **end**
8 **end**
9 ▷ Local-global DV-graph merge
10 **for** *each vertex* $\mathbf{v}_p \in \{\mathcal{P}^i_{local}\} \cup \{\mathcal{P}^j_{global}\}$ **do**
11 **if** *an association exists* **then**
12 Update the vertex in $\{\mathcal{P}^j_{global}\}$;
13 **else if** $\mathbf{v}_p$ *only exists in* $\{\mathcal{P}^j_{local}\}$ **then**
14 Add $\mathbf{v}_p$ to $\{\mathcal{P}^j_{global}\}$ as a new vertex;
15 **else**
16 Remove $\mathbf{v}_p$ from $\{\mathcal{P}^j_{global}\}$ based on voting;
17 **end**
18 **end**
19 **return** Merged DV-graph $\mathcal{G}$;

that make each $\langle \mathbf{p}^i_{topo}, \mathbf{p}^j_{topo} \rangle$ re-connected. The input of this module is the local contour graph $\{\mathcal{P}_{local}\}$. For each movable object $\{\mathcal{P}^i_{mov} \subset \mathcal{P}_{local}\}$, the simulation process is described in Algorithm 1, Line 5-9. Given an arbitrary pair of topological waypoints $\langle \mathbf{p}^i_{topo}, \mathbf{p}^j_{topo} \rangle$ with the estimated push affordance μ of the object, the strategy generation function $\Gamma : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R} \times \mathbf{SE}(\mathbf{2})$ outputs the interaction strategies π along with the estimated manipulation cost $\mathcal{J}$. Because the output does not restrict manipulation actions, general real-time mobile manipulation solutions can be conveniently adopted into this framework with minor changes. The detailed design of the interaction strategies generation is discussed in Section VI-B.

D. Local-Global DV-graph Update

After constructing $\{\mathcal{P}_{local}\}$, we merge the overlapped area of $\{\mathcal{P}_{local}\}$ and $\{\mathcal{P}_{global}\}$ to update the global DV-graph $\mathcal{G}$. The process is described in Algorithm 1, Line 11-19. For each vertex in $\{\mathcal{P}_{local}\}$, we check if there's a close match in $\{\mathcal{P}_{global}\}$ within a certain distance threshold. If so, we update the existing vertex in $\{\mathcal{P}_{global}\}$. If the vertex only exists in $\{\mathcal{P}_{local}\}$, we regard it as a new vertex and add it into the $\{\mathcal{P}_{global}\}$. Conversely, vertices only in $\{\mathcal{P}_{global}\}$, which means they are not observed in the current frame, are not immediately removed. To mitigate sensor noise, we develop a voting mechanism where a vertex is removed from $\{\mathcal{P}_{global}\}$ only after being continuously unobserved for several frames.

VI. Interaction Strategy Generation

In this section, we propose our solution for Problem 2. We first decompose the Problem 2 into a series of sub-problems (Problem 3). For each sub-problem, we conduct

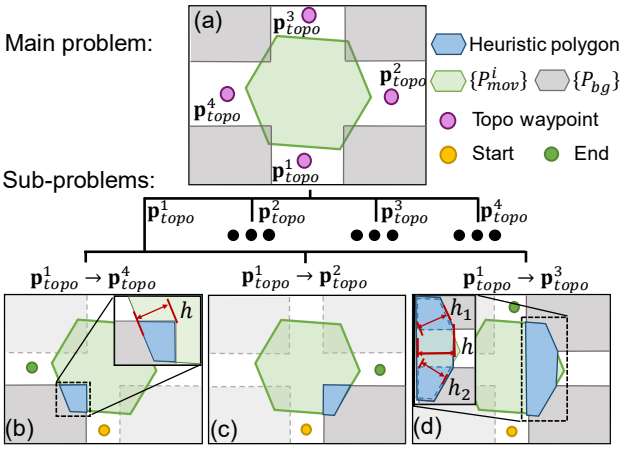

Fig. 4: Illustration of the problem decomposition. (a) The main problem contains $n(n-1)$ sub-problems. (b-d) Three typical sub-problems. The upper-right corner of (b) explains the calculation of the heuristics. The left-upper corner of (d) demonstrates the case when there are multiple heuristic polygons.

a kinodynamic path search for a feasible interaction strategy by expanding motion primitives with a discretized control input space (Sect. VI-B). Our novel design of heuristics is proven to be admissible in general cases and fits well with our sparse polygon-based map representation.

A. Problem Decomposition

As discussed in Sect. V-C, there are $n(n-1)$ local policies to be generated. It is hard to solve via a single search because the state space is large and with multiple sub-goals, and it is hard to design heuristics with an admissibility guarantee. Therefore, we decompose the original problem into $n(n-1)$ sub-problems and solve them separately.

The process of the problem decomposition is illustrated in Fig. 4. Each sub-problem is defined as:

Problem 3: Given the start position $\mathbf{p}^i_{topo}$ and goal position $\mathbf{p}^j_{topo}$, find the most energy-efficient manipulation policy that connects $\mathbf{p}^i_{topo}$ and $\mathbf{p}^j_{topo}$ via a local collision-free path.

The problem is solved by a hybrid A* path search method with a novel heuristic design. It has two advantages: 1) the found interaction strategy, or path, is kinodynamically feasible for the robot to execute, and 2) the designed heuristics are admissible in general cases, which improved the optimality of the path. Technical details can be found in Sect. VI-C.

B. State Transition Model

In this section, we discuss the state transition modeling in the kinodynamic path search algorithm. For simplicity of our system, we use differential vehicles: $v = [v_x, 0], \omega \in \mathbb{R}$, and use push manipulation as our interacting method.

Due to the sensor limitation of mobile robots, our system only infer the object's shape from on-board depth input (where occlusion is inevitable) and physics properties by push attempts. For this reason, we choose stable push [14] as our scheme. It's more robust to the estimation error of object's shape and physics properties compared with other

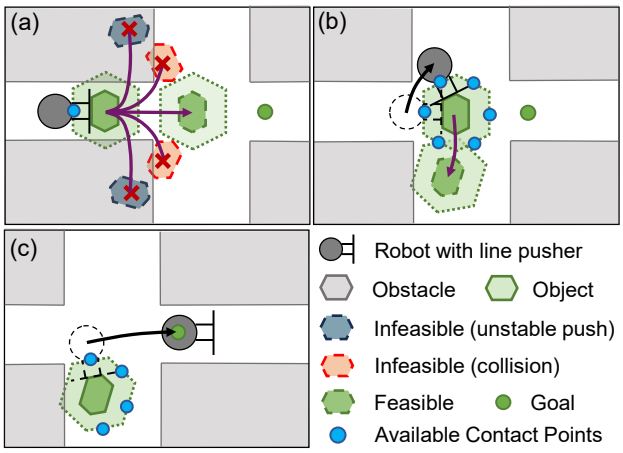

Fig. 5: Hybrid A* push search with contact point switch.

push schemes [15]–[17]. It also simplifies the generation of kinodynamically feasible push strategies for our vehicle.

Given an object with known shape, friction center, contact edge, and the robot to object friction coefficient, we can identify a set of potential stable rotation centers P_s by the procedure "STABLE" in [14]. In our case, the object shapes and physics properties are unknown at first. We estimate the shape of the object in V-A, and start with an initial robot to object friction coefficient k. It is assumed to be uniform so that the friction center aligns with its geometric center. Then, for a certain contact edge (which determines the robot state), we select the vehicle's control inputs which will only result in stable pushes of the object, and map them to the state of objects. Such mapping is trivial because of stable push.

For our case, the full state space $\mathbb{S}'$ of problem 2 is the combination of the robot's and object's states: $\mathbb{S}' \subset \mathbf{SE(2)} \times \mathbf{SE(2)}$. Because of the bijection between the robot state and the object's contact points, the robot's state can be simplified as a set of available contact points $\mathbf{C} \subset \mathbb{R}^2$, shown as blue points in Fig. 3(d). The state space is then simplified to $\mathbb{S} \subset \mathbf{C} \times \mathbf{SE(2)}$. The state vector of a movable object is comprised of object position and orientation $x = [p_x, p_y, \psi]^T$ and available contact points $c = [c^1, c^2, ..., c^n]^T$, where $c^i \in \mathbf{C}$. The state transition model is then derived by: a) applying the differential vehicle model to the movable object and aligning the heading with the normal of the stable push edge. b) restrictig $\frac{\omega}{v_x}$ to let the instantaneous rotation center p fall into P_s. A state transition model whose actions are feasible and transitions are robust is generated. The physics properties will update during the execution of push strategies, which we discuss in VII.B.

C. Hybrid A-Star Path Search

Given a pair of topological waypoints, the problem 3 is solved in two steps. First, find the polygon with the smallest area that hinders the connection of these topological waypoints, demonstrated as blue polygons in Fig. 4(b-d). Here we call it heuristic polygon because it provides heuristics to guide the path search. We represent it as $\{\mathcal{P}_{heu}\} = \{\mathcal{P}^i_{mov}\} \cap \{\mathcal{P}_{bg}\}$. If there are multiple heuristic polygons between the two regions, $\{\mathcal{P}_{heu}\}$ is chosen as the union of the vertex set of all these polygons, as shown in Fig. 4(d).

The heuristic (or the negative reward) of the path search is the minimum distance between parallel lines of the polygon, which is mathematically defined to be the width of the polygon [18]. A hybrid-state astar is then performed with the state and transition model discussed in Sect. VI-B.

D. Completeness and Admissibility

The completeness cannot be theoretically guaranteed because the state space is discredit. However, our method supports variable-resolution state space to improve the completeness guarantee [19] and the practical performance is satisfactory. For optimality, we can prove the admissibility when the heuristic polygon $\{\mathcal{P}^i_{heu}\}$ or the union of all vertices of multiple heuristic polygons $\{\mathcal{P}^{union}_{heu}\}$ are convex.

Lemma 1: h is admissible if $\{\mathcal{P}^i_{heu}\}$ is convex.

According to the definition stated in [18], h is the minimum distance that disengage two convex polygons from contact. Therefore, h is admissible.

Theorem 2: h is admissible if $\{\mathcal{P}^{union}_{heu}\}$ is convex.

Proof: As shown in the upper-left part of Fig. 4(d), if the union is convex, h must satisfy the condition that $h \leq h_1 + h_2$. Because h_1 and h_2 are both admissible for their heuristic polygons, therefore h is admissible for the union.

VII. Interactive Motion Planning and Online Adaptation

A. Global Path Search on DV-graph

To identify the DV-graph $\mathcal{G}$ for the shortest path from $\mathbf{p}_{robot}$ to $\mathbf{p}_{goal}$, the planner first add $\mathbf{p}_{robot}$ and $\mathbf{p}_{goal}$ as two vertices, connecting them to other visible vertices in P^i_{global}. A breadth-first search is then executed on $\mathcal{G}$ to find the shortest path, denoted as a set of waypoints $\mathbf{P}_{path} = \{\mathbf{p}^i_{wp} \in \mathcal{Q} | i \in \mathbb{Z}^+\}$, if it exists. During the navigation among the unknown environment, vertices and directed visibility edges are dynamically updated and stored in $\mathcal{G}$. In the subsequent operations, the DV-graph $\mathcal{G}$ can be loaded as a prior map. When the environment is fully explored, the global planner is capable of efficiently conducting real-time, globally optimal path searches in complex environments ($455m$ path with 1570 vertices in $3ms$, as tested).

B. Path Execution and Adaptable Replan

During the navigation, the robot switches the local planner based on the current path segment it is executing. The working principle is demonstrated in Algorithm 2. The robot employs collision-free planner [20] for regular navigation. Upon nearing a topological waypoint, it switches to interactive planner to execute the planned interaction strategies discussed in Sect. VI. The interactive planner considers the following constraints during the runtime: push stability, collision avoidance and push affordance μ of the robot (e.g., how much weight can it push, the friction coefficient between pusher and slider). We utilize the same method discussed in Sect. VI-B for path replan.

With each sensor frame, the robot utilizes tactile sensing to update the physical properties of the movable obstacle and therefore update the push affordance. The design of our tactile feedback module is based on three principles:

1) If the state of the object doesn't change after maximum push effort, we mark the object as not pushable.
2) If $||\mathcal{J} - \hat{\mathcal{J}}||$ exceeds a threshold, update $\mathcal{J}$ and u_x.

Based on the updated $\mathcal{J}$ and u_x, the planner adaptively replans the interaction primitive or the global path as necessary. After the push execution, the updated $\mathcal{J}$ and π are encoded into the DV-graph $\mathcal{G}$. No matter whether the object is affordable, the proposed method can always adapt to the affordance update and replan the proper strategy accordingly.

Algorithm 2: Path Execution and Adaptation

Input : Next waypoint: $\mathbf{p}^{next}_{path} \in \mathbf{P}_{path}$, start: $\mathbf{p}_{robot}$, goal: $\mathbf{p}_{goal}$, cost: $\mathcal{J}$, affordance: μ

```
while p_robot ≠ p_goal do
    if p_wp^next ∈ p_topo and ||p_wp^next − p_robot|| < r then
        J^, π^, μ^ ← Γ(⟨p_robot, p_wp^next⟩, μ);
        if ||J^ − J|| > τ_thres then
            J ← J^; π ← π^; μ ← μ^;
            Update correlated edges in G based on J^;
            Replan the global path;
        else
            InteractivePlanner(π);
        end
    else
        CollisionFreePlanner(p_wp^next);
end
```

VIII. Experiments

We conduct 10 navigation tasks in simulated environments with different scales, as shown in Fig. 6. For each task, The start and goal positions are chosen based on the criteria that the optimal path length is larger than a threshold. The threshold is $15m$ for the $32m \times 32m$ environment and $80m$ for the $330m \times 270m$ one. The simulated ground vehicle is equipped with a Velodyne Puck Lidar used as the range sensor for perception and a 1-D force sensor for tactile sensing. The framework runs on a laptop with an i7-12700H CPU. We configure the algorithm to update the DV-graph at 2.5Hz and perform path search at each graph update. The spatial resolution is set as $0.15m$. The local layer is a $60 \times 60m$ area with the vehicle in the center.

A. Computational Efficiency Comparison

To demonstrate the computational efficiency of the proposed framework, we compare our algorithm with three classical path planning methods: **A***, **RRT*** and **BIT***, two NAMO methods: **R-NAMO** and **LAMB**, and our previous work **FAR** that uses a V-graph without considering interaction. The experiment results are in Table. I. We used C++ implementation unless explicitly noted in italic fonts. The ground truth (GT) optimal path is the length of the path search by A* on the dense grid map without inflation. The average time consumption is recorded in Table. I if the path

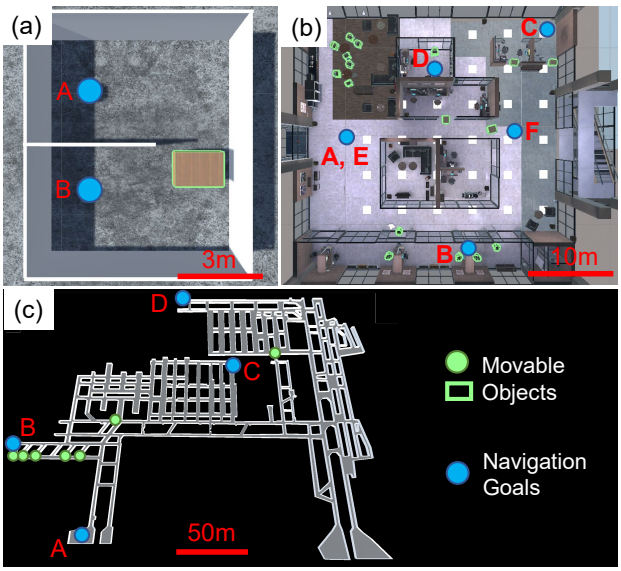

Fig. 6: Environments overview.

search is successful. The time recorded for sampling-based methods represents the duration until the method finds a sub-optimal path whose length is no more than 1.5 times the length of the GT optimal path.

All methods perform similarly in the room environment, considering that the speed of Python can be up to 96.3 times slower than C++ [21]. In the office scenario, the search-based method runs slower due to the increase of the environment scale, while all NAMO methods fail in this scale. Only FAR and our method run in real-time and maintain similar speeds in the tunnel scene. The reasons for the performance of our work and FAR can be attributed to 1) the sparse map representation, making the path search faster in orders of magnitude, and 2) the inherent efficiency of C++.

B. Path Efficiency Comparison

To demonstrate the path efficiency of the proposed system, we conducted this experiment using the same experiment settings as Sect. VIII-A. Table. II presents the results of the experiments. The performance of path efficiency is evaluated by SPL [22]. The table shows that A* performs best in the room scenario because of its mapping density and optimality guarantee. R-NAMO and LAMB have lower SPL in the room with movable obstacles because their manipulation policies take redundant actions. However, our method can plan high-quality manipulation policies because it is admissible in general cases. In Office and Tunnel, with the increasing scale of the environments, the path length of manipulation actions has a lower ratio than the total path length; therefore, our method achieves higher SPL in these scenarios. However, R-NAMO and LAMB fail due to their low scalability, and other methods have to take alternate paths or even fail in some sub-tasks as they cannot conduct interactive navigation, thus resulting in lower SPL.

C. System-level Comparison

To illustrate the potentiality and reliability of our algorithm in field application scenarios, we compare the systematic

TABLE I: Average Search Time in [ms]

Environment	Room (no objects)	Room (with objects)	Office	Tunnel
Optimal Path Length (m)	8.6	8.6	47	455
A*	*2.2/179.0(py)*	-	16.0	363.5
RRT*	0.4	-	**0.4**	325.2
BIT*	**0.17**	-	0.7	1.1e3
R-NAMO	*1.1e3(py)*	*1.2e3(py)*	-	-
LAMB	*7.0e3(py)*	*34.8e3(py)*	-	-
FAR	0.3	-	**0.4**	**1.27**
Ours	0.3	**0.3**	**0.4**	1.32

TABLE II: SPL - Success weighted by Path Length

Environment	Room (no object)	Room (with object)	Office	Tunnel
A*	**0.97**	-	0.41	0.81
RRT*	0.84	-	0.37	0.77
BIT*	0.83	-	0.39	0.80
R-NAMO	0.96	0.58	-	-
LAMB	0.96	0.61	-	-
FAR	0.96	-	0.43	0.79
Ours	0.96	**0.70**	**0.86**	**0.97**

navigation performance with FAR. The reasons for comparing these two algorithms are: 1) the superiority of FAR over other path search methods has been demonstrated in [12], and 2) the offline implementation of R-NAMO and LAMB make them fail to be applicable navigation systems for comparison. We choose Office and Tunnel for test because they are more close to the field applications. The evaluation metrics included travel distance, navigation time, and success rate for various sub-tasks, with both systems configured identically in terms of local map size, resolution, and update frequency.

The results are shown in Fig. 7-8 and Table. III. Both algorithm achieves satisfying performance in terms of success rate. However, due to the lack of interactive navigation ability, FAR planner makes more attempts to finish each sub-task, and sometimes even fail to achieve the goal, e.g. waypoint D in Fig. 6(b). The results in Table. III proves that our algorithm has higher efficiency in different environments.

IX. Limitation and Future Work

In this work, we use the same algorithm for both interactive planning and push manipulation, with objects represented as polygons. However, polygon approximation can introduce errors in push manipulation, such as inaccuracies in contact points and challenges in modeling friction, which affect system robustness. For future work, we will retain polygon representation for planning due to its simplicity and efficiency but aim to incorporate reinforcement learning in manipulation to better handle complex factors like friction.

X. Conclusion

In this paper, we present InteractiveFAR, an interactive navigation system for complex unknown environments cluttered with movable obstacles. Utilizing dynamic DV-graph that integrates interactions during mapping, the system outperforms benchmarks in path searching. The proposed interactive motion planning and adaptive replan framework helps our system manipulate movable obstacles and adjust

TABLE III: System-level Comparison

Environment	Office			Tunnel		
Metrics	Travel Distance (m)	Time (s)	Success Rate	Travel Distance (m)	Time (s)	Success Rate
FAR	232.56	360.34	4/5	1489.63	1911.45	3/3
InteractiveFAR	**109.72**	**242.89**	**5/5**	**1120.91**	**1486.78**	3/3

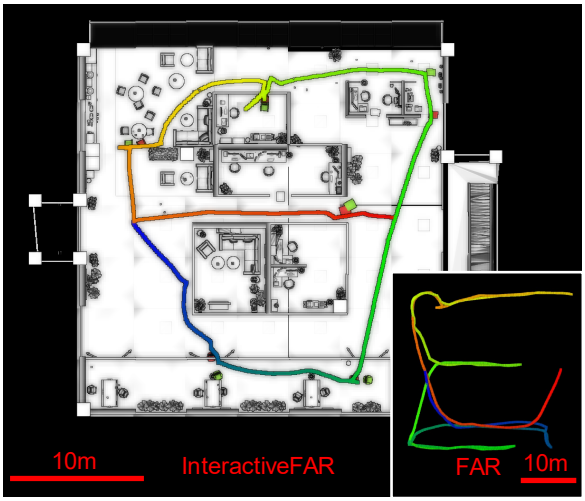

Fig. 7: The resulting map and trajectories of system-level experiment in Office.

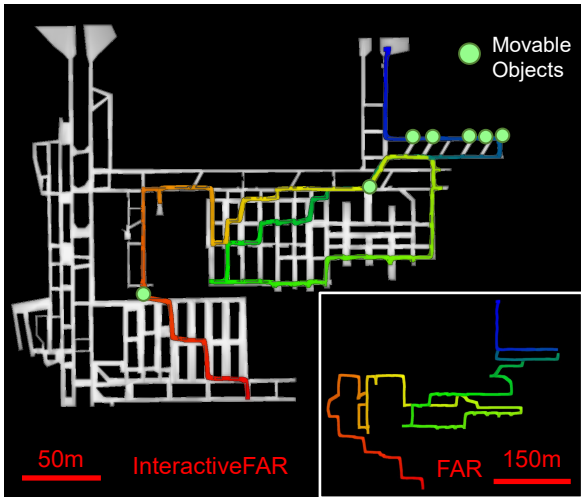

Fig. 8: The resulting map and trajectories of system-level experiment in Tunnel.

strategies in real-time based on new sensor data. Comprehensive experiments and benchmark comparisons validate the efficiency and potential of our system in field applications.

XI. Acknowledgement

The support of NSF under awards OISE 2020624 and BCS 2318255 is greatly acknowledged.

References

[1] C. Cao, H. Zhu, H. Choset, and J. Zhang, "Tare: A hierarchical framework for efficiently exploring complex 3d environments." in *Robotics: Science and Systems*, vol. 5, 2021.

[2] A.-H. Shahidzadeh, S. J. Yoo, P. Mantripragada, C. D. Singh, C. Fermüller, and Y. Aloimonos, "Actexplore: Active tactile exploration on unknown objects," *arXiv preprint arXiv:2310.08745*, 2023.

[3] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu *et al.*, "Swarm of micro flying robots in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm5954, 2022.

[4] G. Wilfong, "Motion planning in the presence of movable obstacles," in *Proceedings of the fourth annual symposium on Computational geometry*, 1988, pp. 279–288.

[5] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1295–1307, 2008.

[6] M. Levihn, M. Stilman, and H. Christensen, "Locally optimal navigation among movable obstacles in unknown environments," in *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2014, pp. 86–91.

[7] J. Muguira-Iturralde, A. Curtis, Y. Du, L. P. Kaelbling, and T. Lozano-Pérez, "Visibility-aware navigation among movable obstacles," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 10 083–10 089.

[8] F. Xia, W. B. Shen, C. Li, P. Kasimbeg, M. E. Tchapmi, A. Toshev, R. Martín-Martín, and S. Savarese, "Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 713–720, 2020.

[9] K.-H. Zeng, L. Weihs, A. Farhadi, and R. Mottaghi, "Pushing it out of the way: Interactive visual navigation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9868–9877.

[10] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[11] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: the international journal for geographic information and geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[12] F. Yang, C. Cao, H. Zhu, J. Oh, and J. Zhang, "Far planner: Fast, attemptable route planner using dynamic visibility update," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 9–16.

[13] A. Rodriguez and M. T. Mason, "Path connectivity of the free space," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1177–1180, 2012.

[14] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *The international journal of robotics research*, vol. 15, no. 6, pp. 533–556, 1996.

[15] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[16] F. R. Hogan, E. R. Grau, and A. Rodriguez, "Reactive planar manipulation with convex hybrid mpc," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 247–253.

[17] N. Doshi, F. R. Hogan, and A. Rodriguez, "Hybrid differential dynamic programming for planar manipulation primitives," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6759–6765.

[18] M. E. Houle and G. T. Toussaint, "Computing the width of a set," in *Proceedings of the first annual symposium on Computational geometry*, 1985, pp. 1–7.

[19] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The international journal of robotics research*, vol. 29, no. 5, pp. 485–501, 2010.

[20] C. Cao, H. Zhu, F. Yang, Y. Xia, H. Choset, J. Oh, and J. Zhang, "Autonomous exploration development environment and the planning algorithms," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 8921–8928.

[21] F. Zehra, M. Javed, D. Khan, and M. Pasha, "Comparative analysis of c++ and python in terms of memory and time. 2020," *Preprints.[Google Scholar]*, 2020.

[22] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva *et al.*, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018.