Minimal Path Violation Problem with Application to Fault Tolerant Motion Planning of Manipulators

Aakriti Upadhyay[1] Mukulika Ghosh[2] Chinwe Ekenna[1]

***Abstract*— Failure of any component in a robotic system during operation is a critical concern, and it is essential to address such incidents promptly. This work investigates a novel technique to recover from failures or changes in the configuration space while avoiding expensive re-computation or re-planning. We propose the Minimal Path Violation (MPV) concept to find the best feasible path with minimal re-configurations. The algorithm ranks pathways based on visibility, expansiveness, and cost. We perform experiments with articulated 3 DOF to 28 DOF robots ranging from serial linkage robots, Kuka YouBots, and PR2 robots. Our results show that our method outperforms existing optimal planners in computation time, total nodes, and path cost while preserving path feasibility in changed configuration space.**

I. Introduction

Designing an autonomous, fault-tolerant manipulator [1], [2] requires identifying configurations that can withstand failures with minimal displacement of constraints while maintaining the manipulator's dependability in the event of partial mechanism failures. Such manipulators are especially useful in complex, confined, and dynamic environments such as disaster areas, nuclear disposal sites, and deep-sea or space exploration. Since these environments are often uncharted and dynamic, positioning robotic manipulators in optimal configurations is critical for high fault tolerance and performance. Choosing a solution path with fewer configurations can make fault tolerance more practical since it requires minor adjustments in the relative configuration when there are changes in the planning space.

Optimal or sub-optimal motion planning for manipulators has been extensively studied [3], [4], [5]. Due to the high dimension of the planning space, sampling-based motion planning (SBMP) [6] algorithms are often preferred for this robotic system. RRT and its variants [7], [8] have demonstrated faster and more computationally viable approaches for solving high-dimensional motion planning problems in robot manipulators [9]. However, finding an optimal path in changing and dynamic planning space using sampling-based planners may result in longer computation times. It increases the computational cost of these planners when addressing failures or any sudden changes in these high-dimensional robotic systems.

When a motion planning attempt fails, the goal is to identify the cause of the failure in order to take the appropriate recovery action [10], [11]. Geometric and topological methods have been used to provide proof of disconnection or infeasibility by decomposing the free space, using alpha shapes and other approximations of obstacle space [12], [13], [14]. However, the focus of this paper is not on finding the cause of failures but on reacting or recovering from the changes in the system that occurred due to failures.

This paper introduces a novel algorithm for near-optimal path planning of a robotic system that focuses on fault tolerance and recovery applications. To reduce the number of re-configurations in finding a feasible path, we introduce the novel Minimal Path Violation (MPV) methodology. MPV ensures that there exists a route that is collision-free and connects to start and goal positions with minimum re-configurations required. Firstly, the algorithm constructs the complete graph of the environment and calculates the coarsely diverse paths from [15], [16]. Secondly, it prioritizes these paths using three ranking measurements, i.e., path cost, the node's visibility, and the edge's expansiveness. Thirdly in the event of any change in the planning space, the proposed algorithm suggests an alternate path without recomputing topological information, thereby saving computational costs. The ranking measures allow the identification of the next best route in case of a path failure, which does not get easily invalidated due to changes in the environment. Figure 1 shows the workflow of the proposed algorithm.

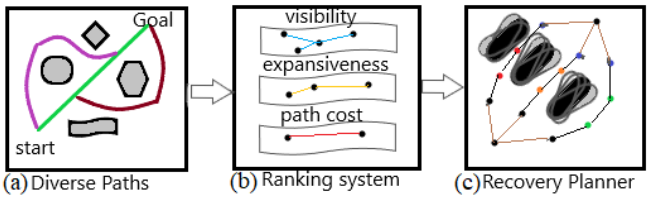


Fig. 1: The algorithm generates diverse paths and ranks them using path cost, node visibility, and edge expansiveness. The points shown in the (c) in red, orange, and green around obstacles are based on the configuration violation detected in the new configuration space.

The proposed contribution involves four main components.

1) We introduce a novel formulation of minimal path violation to compute a feasible path with minimal re-configurations compared to straight-line paths.
2) We introduce a novel ranking measure that prioritizes topologically diverse paths to provide minimal path violation.
3) Our novel path planning algorithm utilizes the ranking measure to find a near-optimal feasible recovery path in the event of failure.
4) The contribution includes the theoretical and empirical proof of the ranking conditions. It helps to establish the efficacy of the proposed algorithm for path planning in dynamic environments, especially in events of failures.

II. Related Work

A robot's placement or configuration can be uniquely defined by a point in a multi-dimensional space, where each

[1]A.Upadhyay and [1]C.Ekenna are with the Department of Computer Science, University at Albany, SUNY. [2] M.Ghosh is with the Department of Computer Science, Missouri State University. {aupadhyay, cekenna}@albany.edu, MGhosh@MissouriState.edu

dimension represents a degree of freedom (DOF). The space consisting of all possible robot configurations, including feasible and unfeasible ones, is called the configuration space ($\mathcal{C}_{space}$) [17]. The configuration of a robot manipulator is typically defined by the set of joint angles, making the joint space a type of configuration space. However, the Cartesian parameters of the robot's end effector do not typically constitute a configuration due to the multiple solutions to the robot's inverse kinematics. In $\mathcal{C}_{space}$, obstacles ($\mathcal{C}_{obst}$) represent unfeasible configurations that could cause collisions, while the complement of $\mathcal{C}_{obst}$, $\mathcal{C}_{free}$, is the set of feasible configurations.

A. Diverse path planning

Planning alternative routes in the presence of obstacles is computationally expensive, which can be a challenge for autonomous robots. Incorporating path diversity into path planning allows for a natural fit with the dynamic planning paradigm. This approach involves generating multiple paths that lead to the same goal, which helps to minimize the need for re-planning in the event of obstacles or motion failures [18], [19]. Recent research has focused on formalizing path diversity and developing effective methods for generating diverse paths during a path planning event [20], [21]. In our previous work [15], we employed a discrete Morse function [16] to identify critical points on the topological roadmap, which provide information about different regions of the $\mathcal{C}_{space}$. Later, our algorithm identified the coarsely diverse paths set incident to these critical points during a single roadmap generation. In this work, we enhance and modify our previous approach to identify and rank routes for fault-tolerant operation in manipulator robots. We improve the results of our method by incorporating node visibility, edge expansiveness, and path cost measures.

B. Fault-tolerant motion planning

The motion planning problem aims to determine a collision-free path, going from a start to a goal configuration, while also avoiding any stationary obstacles that may be in the way. The ability to automatically plan such motions given geometric models of the manipulator and the task is critical for redundant manipulators. However, motion planning failures are common due to the PSPACE completeness of the motion planners and the complexity of the planning environments [22]. To address these issues, researchers use two main approaches: 1) causality analysis, which identifies the causes of planning failures and suggests recovery actions [23]; and 2) avoidance, which leverages domain or historical information to classify planning spaces and problems as easy or difficult [24].

Fabien et al. proposed a polynomial-time algorithm for culprit detection in a constraint network with the relaxed version of the geometric part of the Combined Task and Motion Planning (CTAMP) problem [10]. The algorithm involves constructing a graph of geometric dependencies between the actions of unfeasible symbolic plans to extract sub-sequences of actions that are separately evaluated as potential culprit sub-sequences. Research in [25] proposed a fault-tolerant control scheme for robot manipulators based on active inference of sensory fault. Another notable work presented in [26] focuses on the Minimum Constraint Removal (MCR) problem, which aims to remove the fewest geometric constraints necessary to enable a collision-free path between the start and goal configurations. Hauser presents an SBMP algorithm that clusters connected configurations in each region of the obstacle partition, reducing the number of nodes in the MCR graph. In comparison, we utilize node visibility and edge expansiveness to narrow down the re-configuration search set and prioritize the topological information for faster re-computation of relative paths for fault-tolerant path planning. Our approach can be used in conjunction with the MCR problem, where our algorithm provides the solution by re-using the roadmap information after a constraint is analyzed.

Some other approaches perform fault tolerance planning [27], [28] by classifying and analyzing the planning space problems based on historical or domain knowledge. In this work, we do not focus on learning-based fault detection and instead use the minimum path violation framework to find a feasible path in the changed $\mathcal{C}_{space}$. We define a "change" in the $\mathcal{C}_{space}$ as the inclusion or exclusion of $\mathcal{C}_{obst}$ that invalidates the existing solution.

III. Formal Definitions

In this section, we present Minimal Path Violation (MPV) framework and three ranking conditions to prioritize the diverse paths for fault-tolerance operation.

A. Minimal Path Violation (MPV)

Given a sampled graph G of vertices V and edges E, i.e., $G = \{V, E\}$, such that set V represents the configurations in the closure of $\mathcal{C}_{free}$ and set E represents the collision-free connections between two endpoints $u, v \in V$. Let $\mathbb{P}$ be a set of diverse paths, and $\mathbb{P}_V$ and $\mathbb{P}_E$ define the vertices and edges covered by these paths. We say, a path p is feasible if $\mathbb{P}_V \subseteq V$ and $\mathbb{P}_E \subseteq E$. Let Q' be the set of all violated configurations in changed $\mathcal{C}_{space}$, such that $Q' \subseteq \mathbb{P}_\mathbb{V}$. We define the Minimal Path Violation (MPV) between two endpoints $s, g \in \mathcal{C}_{free}$.

*Definition 1: (**Minimal Path Violation**) A path p' is collision-free and minimal (in the changed $\mathcal{C}_{space}$) if and only if $Q' \cap \mathbb{P}_\mathbb{V}(p') = \emptyset$ and $|\mathbb{P}_\mathbb{V}(p')| - |\mathbb{P}_\mathbb{V}(p)|$ is minimum $\forall p' \in \mathbb{P}$, where $\mathbb{P}_\mathbb{V}(p)$ is the set of vertices covered by p, p is the straight line path between s and g, and $|.|$ represents the cardinality of the set.*

Figure 2 shows a general idea of the MPV problem. We solve the MPV problem in the changed $\mathcal{C}_{space}$ using our ranking measure that assures the minimum number of vertices gets connected to find a route between the start and goal position. Our ranking system takes the node's visibility, edge's expansiveness, and path cost to respond quickly to a changing environment with the next shortest pathway.

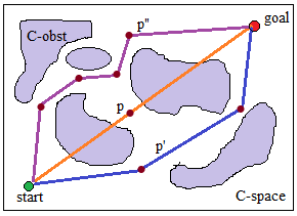


Fig. 2: An illustration of an MPV problem in a 2D $\mathcal{C}_{space}$ where p is an unfeasible straight line path between s and g. Our ranking system solves this problem by prioritizing p' over p'' such that the number of nodes difference between p and p' is minimum while it is also the shortest path.

Consider S as the $\mathcal{C}_{free}$ topology-approximated roadmap with n diverse paths in it, where $S \subseteq G$ and $|\mathbb{P}| = n$. A roadmap S is feasible if the number of valid paths is larger than the number of invalid paths in the new $\mathcal{C}_{space}$, i.e., $(n - t) > t$, where t is the count of the number of invalid paths. We use the bound of MPV, as stated in Theorem 1, to assess the validity of roadmap S.

Theorem 1: Given a roadmap S with m valid configurations and r is the lower bound of intermediate configuration nodes required to connect a path between source S_a and destination S_b in the $\mathcal{C}_{space}$, where $S_a, S_b \in \mathcal{C}_{free}$; $m > r$. Then, the bound for finding the minimal path violation in S is $\frac{r*(m-r)!*r!}{m!}$.

Proof: Let L be the set of configurations of the shortest path p, i.e., $|L| = r$. To invalidate path p in $\mathcal{C}_{space}$, the vertices of L should be arranged such that at least one configuration $c \in L$ is violated or is in collision with $\mathcal{C}_{obst}$. So, the number of permutations of getting a configuration violation that invalidates the path is $P(r, 1) = r$, refer [29]. For m configurations in S, the combination of finding the diverse paths with r configurations is $C(m, r) = \frac{m!}{(m-r)!r!}$, refer [30]. Hence, the bound for finding the minimal path violation for S can be given by $\frac{P(r,1)}{C(m,r)} = \frac{r*(m-r)!*r!}{m!}$. ■

We assess the feasibility of the ranked set during recovery planning (in Alg. 2) using theorem 1. This evaluation aids the algorithm in determining the specific point at which the MPV approach becomes incapable of finding feasible solutions.

B. Ranking Measures

In this section, we discuss our ranking measures to prioritize the diverse paths based on the topology properties of the $\mathcal{C}_{space}$, i.e., path cost, node visibility, and edge expansiveness. We perform these measures on a topology-approximate roadmap defined in our prior work [31], [16]. From this topology map S, we compute the diverse paths set $\mathbb{P}$ and perform a ranking of the routes via the conditions described in Proposition 1 - 3.

*Proposition 1: (**Path Cost**) Taking $\mathbb{P}$ as the set of diverse paths, then a path p is given priority if it has the shortest length compared to the $n - t$ paths.*

Recall that n is the total number of diverse paths in $\mathbb{P}$ and t is the total number of invalid pathways. Here, the general idea of sorting the routes based on their lengths in $\mathbb{P}$ is considered, thus, giving priority to the shortest path with feasibility, i.e., from the shortest path length to the longest. A route becomes the next best alternative if it connects to fewer vertices and edges than the remaining valid paths in the modified $\mathcal{C}_{space}$. □

*Proposition 2: (**Node visibility**) Let K define the limit of maximum visibility. If a vertex $v \in V$ has j connected neighbors where $j \leq K$ and $j \neq 0$. Then, we define the priority of v with the rank score achieved on $K - j$, i.e., the higher the rank score means the highest priority.*

Here, K defines the maximum number of unique neighbors a vertex (or a configuration node) can connect, and j is the number of successful connections made for K connection attempts. The visibility of v depends on the number of unsuccessful connection attempts made and computed by the parameter $l = K - j$ where $l > 0$. The larger the value of l, the lower the visibility of the v in the $\mathcal{C}_{space}$ (existence in narrow regions). We prioritize the low visibility nodes because they are closer to $\mathcal{C}_{obst}$, and the path connecting it will pass at proximity to $\mathcal{C}_{obst}$. Moreover, the chances of finding an alternate route through low-visibility nodes are less than through high-visibility nodes. So, the process of discarding low-visibility nodes when they become invalid and transitioning to higher-visibility nodes during reconfiguration becomes simpler through ranking. As shown in Figure 3, x has a low visibility than y and z. It is, thus, given higher priority than y and then z. □

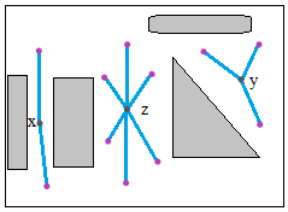


Fig. 3: Visibility ranks of vertices. The vertices with low visibility or lying closer to $\mathcal{C}_{obst}$, such as x, are ranked higher than vertices with high visibility (y and z).

*Proposition 3: (**Edge expansiveness**) Let S_a and S_b be vertices in a roadmap S of the $\mathcal{C}_{free}$ topology-approximation graph G. Suppose the shortest unfeasible path between S_a and S_b in the $\mathcal{C}_{space}$ has length D. If an edge β connects two vertices between S_a and S_b and has length $h \leq D$, then the priority of β is determined by the rank score $D - h$. Specifically, the smaller the rank score, the higher the priority of the edge β.*

We can justify the proposition as follows. First, consider a straight line in $\mathcal{C}_{space}$ connecting S_a and S_b, which need not be collision-free. The collision-free edges between the vertices in the roadmap S determine the safe route for the robot in $\mathcal{C}_{free}$. Suppose the length of edges connecting vertices between S_a and S_b is at most h_{max}. Then, the collision-free edges of length h closer to D will need fewer nodes to form a path than those further away. Hence, longer collision-free edges are more desirable as they reduce the number of nodes in the pathway. This property can help search for the next feasible edge with minimum displacement from the same vertex. In particular, the priority of an edge β connecting vertices between S_a and S_b is based on its rank score $D - h$. Since D is the length of the shortest unfeasible path between S_a and S_b, edges with smaller lengths $h < h_{max}$ have higher rank scores and lower priority.

To illustrate, consider Figure 4, where the length of the shortest unfeasible path between S_a and S_b is D, and h_U is the edge between S_a and S_b, h_V is between q to S_b, and h_W is between r to S_b along paths U, V, and W, respectively. Then, during ranking, h_U has higher priority than h_V and h_W, and the rank score is $D - h_U$. Also, in general, D always takes length as h_U between a start and goal position. Therefore, the edge expansiveness property helps efficiently find a feasible path between two vertices in the roadmap by prioritizing longer, collision-free edges. □

When ranking paths, the priority sequence is determined by path cost followed by node visibility and edge expansiveness.

IV. Algorithm Development

To overcome/recover from the faulty scenario, we perform two associated steps. First, we apply Propositions 1 - 3 to rank our feasible coarsely diverse paths. Second, we use the ranking information to find a valid pathway in the changed

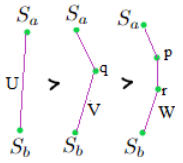


Fig. 4: Ranking of edge expansiveness. It prioritizes longer collision-free edges in the paths, such as U, over short edges in V and W. The symbol $>$ refers to the "greater than" notation.

$\mathcal{C}_{space}$ realizing MPV framework. These two algorithms are performed as the post-processing steps to the already approximated $\mathcal{C}_{free}$-topology graph. [31], [16].

A. Ranking diverse paths

Algorithm 1 aims to rank and return a set of coarsely diverse paths for a given dense graph G, with the aim of solving a query from start to goal position. The algorithm uses a combination of topological and geometric information abstracted from G, as provided by methods introduced in previous works [31], [16].

To compute the diverse paths, the algorithm employs the *GetPath* method (lines 2-10), which generates a set $\mathbb{P}$ of n-distinct paths in S, where the parameter n controls the number of pathways to identify. Next, the algorithm computes the visibility of the configuration nodes in S (from Proposition 2), using the *CountNeighbors* method (lines 11-17). The visibility information prioritizes the nodes that are more likely to be part of high-quality paths in q.

To further prioritize the high-quality paths, the algorithm creates a priority queue e of collision-free edges with longer lengths (as per Proposition 3) in lines 18-21. The algorithm then sorts the set of diverse paths in $\mathbb{P}$ based on their path lengths (as per Proposition 1), using the *Sort* method (line 24). The paths with the lowest cost are ranked first, with ties broken by giving importance to the number of high-priority nodes and then high-priority edges (lines 25-26).

The algorithm outputs a set R of ranked paths that get used in algorithm 2 for solving queries between the same start and goal position in changed $\mathcal{C}_{space}$. Overall, Algorithm 1 combines our various methods and techniques [31], [16], [15] to efficiently rank and generate a diverse set of high-quality paths from a given dense graph.

B. Minimal path violation planning

Algorithm 2 considers the ranked paths set $R \subset S$ and the environment map X of the new $\mathcal{C}_{space}$ as the input, where $dim(S) \neq dim(X)$. Here, dim calculates the dimension of the space. The $Projection$ method maps the configuration nodes of $dim(S)$ to $dim(X)$ using transformation matrix I in lines 3-6.

$$I_{i*j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}; \quad i = dim(S), j = dim(X). \tag{1}$$

Using the map X, the algorithm evaluates the validity of the paths in the set R in line 7. The pathway is pruned from R if the current shortest path is invalid due to new changes in the $\mathcal{C}_{space}$, and the adjacent shortest route from R gets selected for validation in lines 8-10. When the number of valid nodes decreases in $R \subset S$, the number of invalid paths increases, thus, making the roadmap S unfeasible, i.e., $R \rightarrow null$. The increase in the probability of MPV (from theorem 1) for R decreases the chances of finding a feasible solution in X which tends towards 0 (line 10). The process continues until a near-optimal pathway is found in X, else the algorithm calls a complete planner (like PRM [32], RRT, e.t.c.) to find a path due to the discretization of the space.

Algorithm 1 Ranking n coarsely diverse paths

Input: G: A dense graph comprising of vertices set V and edges set E where $G = \{V, E\}$, Q: query to be solved from start to goal position, $\mathbb{P}$: set of n-distinct paths, p: vector array of path vertices, n: number of paths to be computed, K: node-reachability parameter, D: distance b/w start and goal position, R: set of ranked paths.

1: Let $i = 0$, $\mathbb{P} \leftarrow NULL$, $p = 0$.
2: $T = GetTopologyGraph(G)$; ◁ Refer [31]
3: $M = GetCriticalPoints(T)$; ◁ Refer [16]
4: $S = T \bigcup M$
5: **while** $i < n$ **do**
6: Q = false
7: $p = GetPath(S)$; ◁ Refer [15]
8: **if** Q = true **then**
9: $\mathbb{P} = \mathbb{P} \bigsqcup p$
10: i = $|\mathbb{P}|$
11: $q \leftarrow NULL$
12: **for all** nodes $x \in S(V)$ **do**
13: $j = CountNeighbors(x, S(V))$; ◁ Proposition 2
14: **if** $j \neq K$ **and** $K - j > 0$ **then**
15: $PushFront(q, x)$;
16: **else if** $j == K$ **then**
17: $PushBack(q, x)$;
18: $e \leftarrow NULL$
19: **for all** edges $y \in S(E)$ **do**
20: $h = D - length(y)$; ◁ Proposition 3
21: $e.push(h, y)$;
22: $r \leftarrow NULL$
23: Sort($\mathbb{P}$); ◁ Proposition 1
24: **for all** $p \in \mathbb{P}$ **do**
25: $r = Rank(p, q, e)$;
26: $R = R \bigcup r$
27: **return** R

Algorithm 2 Recovery path planner

Input: X: a new $\mathcal{C}_{space}$, S_a: start position, S_b: goal position, p: vector array of path vertices, R: set of ranked paths, Γ: a complete planner.

1: $x = dim(X)$;
2: $R = GetRankedPaths()$; ◁ Algorithm 1
3: $S_a \leftarrow Projection(S_a, x)$; $S_b \leftarrow Projection(S_b, x)$
4: **while** S_a and S_b not connected **do**
5: **for all** $p \in R$ **do**
6: $p \leftarrow Projection(p, x)$;
7: **if** p is not valid in X **then**
8: $remove(R, p)$;
9: **if** R not null **then**
10: $p = SelectNextPath(R, X)$; ◁ Theorem 1
11: **else**
12: Calls $\Gamma(S_a, S_b)$
13: **else**
14: Connect p with S_a and S_b
15: **return** p

Using Algorithm 1 to rank our diverse path and Algorithm 2 with MPV for recovery, we reduce the re-planning time and computation cost that we evaluate in Section VI.

V. Experimental Setup

The experiments were executed on a Dell Alienware Aurora desktop machine running Ubuntu 20.4 LTS operating system, and the algorithms were implemented in C++ language, using motion planning library [33]. We used the RAPID [34] collision detection method during the sampling, connection, and query stages and used the brute force k-closest neighbor finding technique [35], the Euclidean distance metric method, and a straight line local planner for sampling and connection stages. We performed experiments

in 3 environments with robots (articulated linkage, Kuka Youbot, and PR2) ranging from 2 to 28 DOF in Cluttered, Pick and Place, and Object displacement environments, respectively. Figure 5 show the start and goal positions in red and blue color.

VI. Results

In this section, we discuss the performance of our method in the modified $\mathcal{C}_{space}$ and compare its results with optimal planners, like LazyPRM* [36], RRT*-Connect [7], and Informed RRT*-Connect [8]. Here, we take K = 5, whereas D gets calculated empirically using a distance function within the algorithm for ranking conditions. We performed a total of 225 executions to generate roadmaps for each planner in all three environments, and the results were averaged over 15 runs to get standard deviation values. We used PRM [32] sampling strategy to sample the configurations in the environment.

A. Solving MPV problem in changed $\mathcal{C}_{space}$

1) Prioritising coarsely-diverse paths: We generate topology maps and diverse paths for all 3 environments using 3 DOF articulated linkage robot, 10 DOF Kuka YouBot, and 28 DOF PR2 robot (two arms), respectively. Table I shows the size of the rank set, the range of path cost, nodes, and edges for the coarsely diverse paths derived from Algorithm 1. This table gives the information needed for MPV, including the minimum and the maximum number of nodes/edges present in the paths in these environments.

TABLE I: Ranked coarsely diverse paths

Environments	Total paths	Cost	N_{nodes}	N_{edges}
Cluttered	84	280-546	10-36	14-22
Pick and Place	3	1584-1901	30-96	38-52
Object displacement	2	504-630	12-18	11-17

2) Fault-tolerant planning in changed $\mathcal{C}_{space}$: Initially, we generate and rank various paths for robots with 3, 10, and 28 degrees of freedom (DOF) in their respective environments. We then introduce dynamic behavior in these environments by modifying the obstacles' positions, constraining the manipulator's joint angles, or limiting the query evaluation time, causing the existing shortest paths to become unfeasible and the planner to fail to identify the same route. Consequently, our approach identifies an alternative pathway with fewer DOF to serve as a recovery measure in these environments. The aim is to find a feasible fault-tolerant solution by reducing the DOF of the robot in consideration (a subset of the original problem) for the detected fault in the initial environment.

a) Cluttered environment: We add two new obstacles and plan a path using a 2 DOF articulated linkage robot to recover from faulty $\mathcal{C}_{space}$. The placement of two recent objects invalidates the current shortest path from Figure 5a, and as a result, our planner looks for the next possible pathway to connect start and goal positions as shown in Figure 5d.

b) Pick and Place environment: We add one new obstacle and restrict the joint angle of one of the links (introduces extra computation time). The initial shortest path of 10 DOF Kuka YouBot is shown in Figure 5b and the fault-tolerant pathway planned using 7 DOF Kuka YouBot is shown in Figure 5e.

c) Object displacement environment: We restrict the forearm movement of the left hand and limit the query evaluation to 10 seconds. The faulty scenario of the PR2 robot is captured in Figure 5c, where it is unable to find a path due to restricted movement of the left hand, and Figure 5f shows the recovery path planned using 14 DOF PR2 robot (one arm). In the changed $\mathcal{C}_{space}$, the right-hand picks both the sticks, which were initially picked separately by each hand. Hence, we have two paths planned for the right hand that performs one stick displacement initially and two sticks displacement for the changed $\mathcal{C}_{space}$.

Table II shows the number of configuration nodes in the initial shortest path and the new shortest path in changed $\mathcal{C}_{space}$ at the time of recovery planning.

TABLE II: MPV solution in new $\mathcal{C}_{space}$

Environments	Initial Path		New Path	
	Nodes	Cost	Nodes	Cost
Cluttered	$10^{\pm 10.76}$	$280^{\pm 2.79}$	$13^{\pm 9.45}$	$286^{\pm 3.4}$
Pick and Place	$30^{\pm 5.11}$	$1584^{\pm 11.23}$	$36^{\pm 2.76}$	$1848^{\pm 9.83}$
Object displacement	$12^{\pm 1.34}$	$504^{\pm 1.0}$	$18^{\pm 1.98}$	$630^{\pm 0.67}$

From Tables I and II, we observe that the re-configuration of nodes during recovery planning maintains the minimality of the MPV problem. Our method selects the next best route within the ranked paths with a minimum re-configuration of 2 to 6 nodes in all 3 environments. Hence, we can conclude that our method efficiently solved the MPV problem to find a valid path by using the ranking measure that guaranteed minimum node re-configuration from the invalid pathways.

Table III shows the displacement distance between the initial and re-planned paths in the changed $\mathcal{C}_{space}$. We compare the results of our method with optimal planners LazyPRM*, RRT*-Connect, and Informed RRT*-Connect. We use the Hausdorff distance [37] to measure the distance between two paths. We observed that our method identifies a feasible pathway in new $\mathcal{C}_{space}$ with the smallest displacement distance between initial and re-planned routes compared to the paths generated by other methods in all 3 environments.

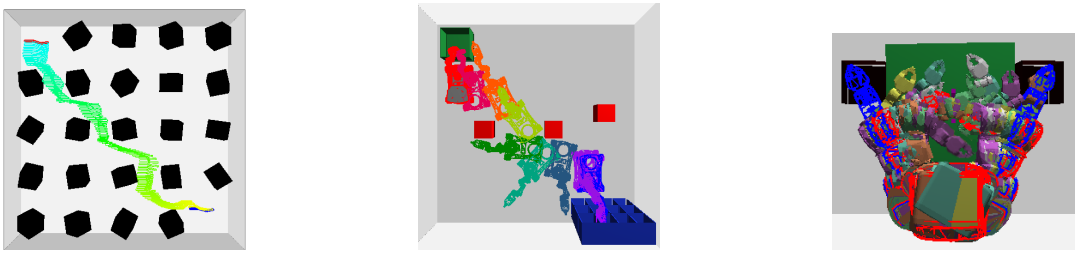
We conclude that using the MPV probability and ranked diverse paths information can aid in finding the next feasible solution with minimum re-configurations.

B. Analysing the Recovery Solution

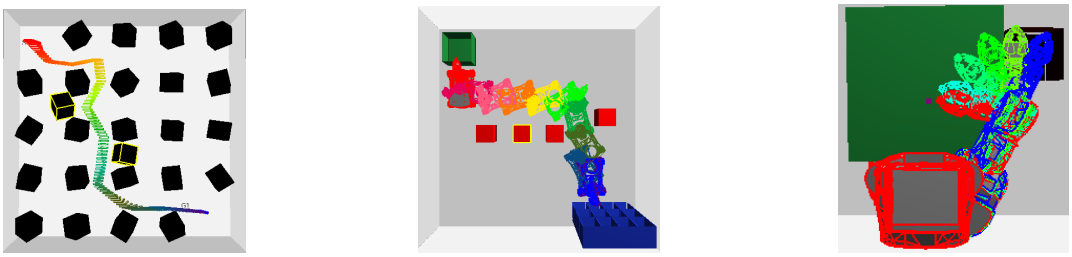
The goal is to ensure a nearly optimal solution while recovering from a faulty scenario. We evaluate our recovery solution against optimal planners in time, cost, and nodes to analyze resource consumption. Using Algorithm 1 and 2, we compute diverse paths and re-use information across three environments through a ranking system. Our planner's computation time includes roadmap generation, ranking, and recovery evaluation. We compare our results with LazyPRM*, RRT*-Connect, and Informed RRT*-Connect, in initial and changed $\mathcal{C}_{space}$, taking average values for planned and re-planned roadmap nodes and path cost. Our method, as seen in Figure 6, outperforms in computation time and path cost without the need to re-generate a roadmap while it generates more nodes than RRT*-Connect and Informed RRT*-Connect due to building a complete graph of $\mathcal{C}_{space}$. LazyPRM* failed to find a path for the PR2 robot due to query resolution, while Informed RRT*-Connect failed to provide valid configurations within the ellipsoidal curve for Kuka YouBot and PR2 robots (computed from [38]).

TABLE III: Distance b/w initial and re-planned paths

Environments	Our Approach	LazyPRM*	RRT*-Connect	Informed RRT*-Connect
Cluttered	$1.07^{\pm 0.08}$	$2.68^{\pm 1.06}$	$2.02^{\pm 0.95}$	$1.27^{\pm 0.27}$
Pick and Place	$13.39^{\pm 0.14}$	$18.85^{\pm 1.99}$	$16.63^{\pm 1.32}$	DNF
Object displacement	$0.01^{\pm 0.007}$	DNF	$0.08^{\pm 2.65}$	DNF

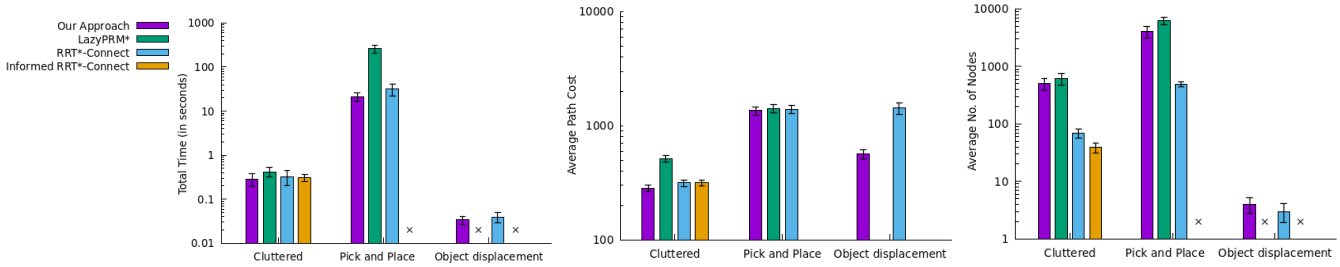


(a) Cluttered (b) Pick and Place (c) Object displacement

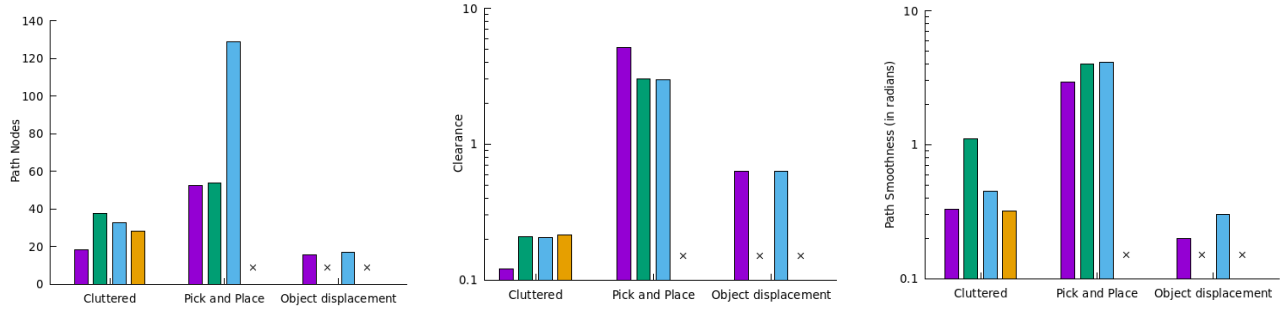


(d) (e) (f)

Fig. 5: Figures (a) and (b) show the initial paths for 3 DOF articulated linkage robot and 10 DOF Kuka YouBot, respectively. Figure (c) shows the faulty scenario for the PR2 robot with a restricted left hand. Figures (d), (e), and (f) show fault-tolerant paths for 2 DOF articulated linkage robot, 7 DOF Kuka YouBot, and 14 DOF PR2 robot, respectively.



(a) **Computation Time** (b) **Path Cost** (c) **Total Nodes**



(d) **Path Nodes** (e) **Minimum Clearance** (f) **Path Smoothness**

Fig. 6: The computation time takes into account the total time taken to compute a path in the initial and the new $\mathcal{C}_{space}$. We take the average of planned and re-planned roadmap values for the path cost, the total number of nodes, clearance, smoothness, and path nodes. The "x" in the plots refers to DNF (did not finish).

a) Path Metric Evaluation: We also evaluate the path metrics of the initial and new paths by comparing the average number of nodes in the routes, the average minimum clearance from the $\mathcal{C}_{obst}$, and the deviation of the new pathway from the initial path using smoothness measure (in radians). Figures 6d and 6f show that our method-planned optimal pathways have fewer nodes and minor deviations in all environments compared to other planners. In Figure 6e,

we noticed a smaller clearance in the Cluttered environment and a higher clearance value for Kuka YouBot and PR2 robot for our planner, which shows its ability to adapt to safety measures in high-dimensional $\mathcal{C}_{space}$.

Overall, we saw a noticeable enhancement in the re-planning time and computation cost for fault-tolerant scenarios. It is worth noting that the complexity of the environment or frequent displacement of $\mathcal{C}_{obst}$ can increase the likelihood of MPV bound and its vulnerability to failures.

VII. Conclusion

This work proposes a novel algorithm to find a solution for fault tolerance and recovery in dynamic planning environments. We introduce the Minimal Path Violation (MPV) concept to efficiently find a feasible solution with minimal re-configurations in the changed $\mathcal{C}_{space}$. Our approach utilizes ranking measures based on node visibility, edge expansiveness, and path cost to swiftly adapt to environmental changes and minimize the need for re-planning. Our results demonstrate that by reducing re-planning time and limiting validity checks to ranked pathways, our method surpasses the optimal methods in fault-tolerant scenarios. Future work includes comprehensive comparisons with other fault-tolerant planning methods and practical applications on physical robotic systems.

References

[1] A. A. Maciejewski, "Fault tolerant properties of kinematically redundant manipulators," in *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE, 1990, pp. 638–642.

[2] H. Abdi and S. Nahavandi, "Well-conditioned configurations of fault-tolerant manipulators," *Robotics and autonomous systems*, vol. 60, no. 2, pp. 242–251, 2012.

[3] M. Kirćanski and M. Vukobratović, "Trajectory planning for redundant manipulators in the presence of obstacles," in *Theory and Practice of Robots and Manipulators*. Springer, 1985, pp. 57–63.

[4] M. d. G. Marcos, J. Machado, and T.-P. Azevedo-Perdicoúlis, "An evolutionary approach for the motion planning of redundant and hyper-redundant manipulators," *Nonlinear Dynamics*, vol. 60, no. 1, pp. 115–129, 2010.

[5] M. Shahabi, H. Ghariblu, and M. Beschi, "Comparison of different sample-based motion planning methods in redundant robotic manipulators," *Robotica*, pp. 1–16, 2022.

[6] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[7] S. Klemm, J. Oberländer, A. Hermann, A. Roennau, T. Schamm, J. M. Zollner, and R. Dillmann, "RRT-connect: Faster, asymptotically optimal motion planning," in *2015 IEEE international conference on robotics and biomimetics (ROBIO)*. IEEE, 2015, pp. 1670–1677.

[8] R. Mashayekhi, M. Y. I. Idris, M. H. Anisi, I. Ahmedy, and I. Ali, "Informed rrt*-connect: An asymptotically optimal single-query path planning method," *IEEE Access*, vol. 8, pp. 19 842–19 852, 2020.

[9] S. Thakar, P. Rajendran, A. M. Kabir, and S. K. Gupta, "Manipulator motion planning for part pickup and transport operations from a moving base," *IEEE Transactions on Automation Science and Engineering*, 2020.

[10] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *The International Journal of Robotics Research*, vol. 35, no. 8, pp. 890–927, 2016.

[11] M. Brandão, G. Canal, S. Krivić, and D. Magazzeni, "Towards providing explanations for robot motion planning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 3927–3933.

[12] Z. McCarthy, T. Bretl, and S. Hutchinson, "Proving path non-existence using sampling and alpha shapes," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 2563–2569.

[13] S. Li and N. T. Dantam, "Towards general infeasibility proofs in motion planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 6704–6710.

[14] A. Varava, J. F. Carvalho, F. T. Pokorny, and D. Kragic, "Caging and path non-existence: A deterministic sampling-based verification algorithm," in *Robotics Research*, N. M. Amato, G. Hager, S. Thomas, and M. Torres-Torriti, Eds. Cham: Springer International Publishing, 2020, pp. 589–604.

[15] A. Upadhyay, B. Goldfarb, and C. Ekenna, "A topological approach to finding coarsely diverse paths," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 6552–6557.

[16] A. Upadhyay, B. Goldfarb, W. Wang, and C. Ekenna, "A new application of discrete Morse theory to optimizing safe motion planning paths," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2022, pp. 18–35.

[17] T. Lozano-Perez, "Spatial planning: A configuration space approach," *Computers, IEEE Transactions on*, vol. 100, no. 2, pp. 108–120, 1983.

[18] R. A. Knepper and M. T. Mason, "Path diversity is only part of the problem," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 3224–3229.

[19] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k shortest paths with diversity," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 3, pp. 488–502, 2017.

[20] T. Chondrogiannis, P. Bouros, J. Gamper, U. Leser, and D. B. Blumenthal, "Finding k-dissimilar paths with minimum collective length," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2018, pp. 404–407.

[21] V. Vonásek, R. Pěnička, and B. Kozlíková, "Computing multiple guiding paths for sampling-based motion planning," in *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, 2019, pp. 374–381.

[22] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[23] A. A. Almarkhi, A. A. Maciejewski, and E. K. Chong, "An algorithm to design redundant manipulators of optimally fault-tolerant kinematic structure," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4727–4734, 2020.

[24] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, "A survey of robot manipulation in contact," *Robotics and Autonomous Systems*, vol. 156, p. 104224, 2022.

[25] M. Baioumy, C. Pezzato, C. H. Corbato, N. Hawes, and R. Ferrari, "Towards stochastic fault-tolerant control using precision learning and active inference," in *Machine Learning and Principles and Practice of Knowledge Discovery in Databases: International Workshops of ECML PKDD 2021, Virtual Event, September 13-17, 2021, Proceedings, Part I*. Springer, 2022, pp. 681–691.

[26] K. Hauser, "The minimum constraint removal problem with three robotics applications," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 5–17, 2014.

[27] F. Piltan, A. E. Prosvirin, M. Sohaib, B. Saldivar, and J.-M. Kim, "An SVM-based neural adaptive variable structure observer for fault diagnosis and fault-tolerant control of a robot manipulator," *Applied Sciences*, vol. 10, no. 4, p. 1344, 2020.

[28] Z. Yan, J. Tan, B. Liang, H. Liu, and J. Yang, "Active fault-tolerant control integrated with reinforcement learning application to robotic manipulator," in *2022 American Control Conference (ACC)*. IEEE, 2022, pp. 2656–2662.

[29] Wikipedia, "Permutation," https://en.wikipedia.org/wiki/Permutation, 2023, accessed: 2023-02-18.

[30] Wikipedia, "Combination," https://en.wikipedia.org/wiki/Combination, 2023, accessed: 2023-02-18.

[31] A. Upadhyay, W. Wang, and C. Ekenna, "Approximating c free space topology by constructing vietoris-rips complex," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2517–2523.

[32] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, August 1996.

[33] P. Lab, "Parasol planning library," https://github.com/parasol-ppl/ppl, 2022.

[34] S. Gottschalk, M. Lin, and D. Manocha, "OBB-tree: A hierarchical structure for rapid interference detection," in *Proc. ACM SIGGRAPH*, 1996, pp. 171–180.

[35] T. McMahon, S. Jacobs, B. Boyd, L. Tapia, and N. M. Amato, "Evaluation of the k-closest neighbor selection strategy for PRM construction," Texas A&M, College Station Tx., Technical Report TR12-002, 2011.

[36] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 2951–2957.

[37] Wikipedia, "Hausdorff distance," 2023, accessed: 2023-02-21. [Online]. Available: https://en.wikipedia.org/wiki/Hausdorff_distance

[38] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2997–3004.