IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 29 November 2022, accepted 1 January 2023, date of publication 10 January 2023, date of current version 13 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3235652

RESEARCH ARTICLE

Anytime Informed Multi-Path Replanning Strategy for Complex Environments

CESARE TONOLA[1,2], MARCO FARONI[3], MANUEL BESCHI[1,2], (Member, IEEE), AND NICOLA PEDROCCHI[2]

[1]Dipartimento di Ingegneria Meccanica e Industriale, University of Brescia, 25123 Brescia, Italy
[2]Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing of the National Research Council of Italy (CNR-STIIMA), 20133 Milan, Italy
[3]Department of Robotics, University of Michigan, Ann Arbor, MI 48109, USA

Corresponding author: Cesare Tonola (c.tonola001@unibs.it)

This work was supported in part by Safe and effective HumAn-Robot coopEration toWards a better cOmpetiveness on cuRrent automation lacK manufacturing processes (ShareWork Project) through Horizon 2020 (H2020), European Commission, under Grant 820807.

ABSTRACT In many real-world applications (*e.g.*, human-robot collaboration), the environment changes rapidly, and the intended path may become invalid due to moving obstacles. In these situations, the robot should quickly find a new path to reach the goal, possibly without stopping. Planning from scratch or repairing the current graph can be too expensive and time-consuming. This paper proposes MARS, a sampling-based **M**ulti-p**A**th **R**eplanning **S**trategy that enables a robot to move in dynamic environments with unpredictable obstacles. The novelty of the method is the exploitation of a set of precomputed paths to compute a new solution in a few hundred milliseconds when an obstacle obstructs the robot's path. The algorithm enhances the search speed by using informed sampling, builds a directed graph to reuse results from previous replanning iterations, and improves the current solution in an anytime fashion to make the robot responsive to environmental changes. In addition, the paper presents a multithread architecture, applicable to several replanning algorithms, to handle the execution of the robot's trajectory with continuous replanning and the collision checking of the traversed path. The paper compares state-of-the-art sampling-based path-replanning algorithms in complex and high-dimensional scenarios, showing that MARS is superior in terms of success rate and quality of solutions found. An open-source ROS-compatible implementation of the algorithm is also provided.

INDEX TERMS Anytime search, dynamic environments, informed planning, motion planning, path replanning, sampling-based algorithms.

I. INTRODUCTION

Moving a robot in the real world presents several challenges, including adapting the robot's motion to the dynamic nature of the environment. This problem has become very important in recent years with the spread of mobile robotics [1], social robotics [2], and human-robot collaboration (HRC) in industrial settings [3]. Path planners usually plan a path from a start point to a goal point, considering only static obstacles. However, unstructured environments may have moving and unforeseen obstructions that invalidate the initially calculated trajectory. In these cases, path replanning algorithms are needed to enable the robot to react rapidly to environmental changes, quickly correcting the robot's route to avoid collisions and reach the goal safely. An example of a possible application is HRC, i.e., workspace sharing between humans and robots. Typically, the robot's speed is modulated according to the distance from the operator to avoid collisions.

The associate editor coordinating the review of this manuscript and approving it for publication was Zheng Chen.

A path replanning algorithm prevents the robot from getting stuck when the operator obstructs the robot's path for a long time.

This work proposes a sampling-based path replanning strategy that exploits multiple pre-computed paths to repair and optimize the solution over time. The method combines an anytime approach and admissible informed sampling [4] to be fast, even in complex, high-dimensional scenarios. The complexity of the search is reduced by connecting the current path to one of the available collision-free paths. The algorithm uses a heuristic to determine which nodes of the other available paths to connect to; informed sampling, subtrees reuse, and a directed graph allows for fewer calculations and a higher solutions convergence rate.

A. RELATED WORKS

Several strategies have been developed over the years to modify the robot's path on the fly.

Graph-based replanning algorithms usually derive from A* [5], a well-known algorithm to find the optimal path in a graph. For example, Lifelong Planning A* (LPA*) [6] is a modified version of A* that repeatedly finds the shortest paths from a given start node to a given goal node as the edge costs of a graph change or vertices are added or deleted. Variants of the algorithm exist [7], [8], [9], but these approaches suffer from the *curse of dimensionality*. Consequently, these algorithms are better suited for small-dimensional problems, such as planning for mobile robots.

Potential fields also are often investigated [10], [11], [12]. In [10], a force that depends on the distance between the robot and the obstacles deforms an initial trajectory; a subsequent force seeks to restore the trajectory. Local minima are a potential drawback of this approach. There may not be a fix if the environmental change results in the disappearance of a passage (*i.e.*, this approach is not complete).

Reinforcement Learning (RL) is an emerging approach for online planners [13], [14]. These strategies are responsive and efficient in the environment they have been trained in. Still, they are not suitable for unfamiliar environments and completely unpredictable obstacles.

Sampling-based replanning algorithms are the most popular planners [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]. Usually, these algorithms are variants of the Rapidly-exploring Random Tree (RRT) algorithm [25], its optimal counterpart RRT* [26], or the Probabilistic RoadMap (PRM) algorithm [27]. Random sampling avoids the *a priori* discretization of the search space. For example, Execution-extended RRT (ERRT) [15] searches for a new path to the goal alternating the sampling of new states and *cached waypoints*. The rationale is that the real-world changes, but the changes are usually small. As a result, the path initially found can be a guideline for finding a new one. Although ERRT proves to be faster than RRT in finding the new solution, there is little information it reuses from previous planning queries. Dynamic RRT (DRRT) [16] takes a different approach, reusing the valid branches of the RRT. When an obstacle obstructs the current path, the algorithm prunes the tree and starts its construction to reach the goal again. This approach involves an initial overhead because, firstly, the algorithm must verify all tree branches. Furthermore, the tree is re-built using RRT so that the new path will be far from optimality. Efficient Bias-Goal factor RRT (EBG-RRT) [21] modifies the two aforementioned methods using an efficient and optimal waypoint cache to connect the valid part of the tree to the path beyond the obstacle.

Many sampling-based path planners combine *anytime searches* [28], [29], and *adaptive sampling strategies* [4] to compute suitable solutions quickly and rapidly refine them, and, therefore, to meet the needs of real-world applications. For example, Anytime Dynamic RRT [17] calls a DRRT when an obstacle stops the current path and uses an Anytime RRT [29] to enhance the solution. *Informed sampling* can speed up the search for better solutions by shrinking the sampling space with an estimation of the *omniscient set*, *i.e.*, the set of samples that can improve the solution. Sampling the *informed set* allows considering only states with a non-null probability of improving the path [4], [30].

Among other works relevant to ours, [31] uses RRT* to repair the path given a potential collision. The nodes around the obstacle are rewired to make the node corresponding to the robot position as their parent node, and new states are sampled to find a path to the goal. Reconfigurable Random Forest (RRF) [32] is multi-query and preserves portions of the tree that are disconnected by the appearance of a new obstacle. The algorithm maintains a forest of disconnected RRTs rooted in different locations and continuously tries to connect them. Multipartite RRT [19] combines RRF, DRRT, and ERRT to maintain a forest of disconnected RRTs and repair the path to the goal. [18] provides the Multiple Parallel RRTs (MPRRT) algorithm, which continuously executes independent RRTs on each available processor core. Dynamic Roadmap (DRM) [23] adapts a PRM [27] to environmental changes by decomposing the workspace into cells and detecting the edges and nodes of the roadmap that affect each cell occupied by obstacles and then searches the graph again. Time dimension can be added to the graph, and the estimation of the movement of obstacles, if available, can be used to predict collisions [33]. Furthermore, it is possible to check vertices and edges inside a time horizon while delaying the examination of those outside it by representing obstacles as time-space volumes [34], [35]. Temporal PRM (T-PRM) [24] exploits obstacle motion prediction to incorporate into the nodes of a PRM the information about the time intervals with no collisions. Unlikely, the algorithm assumes a constant speed for the obstacles.

These methods' performance deteriorates as the search space's complexity and dimensionality increase. When the robot's path is obstructed multiple times, computing new trees from scratch each time or performing tree pruning can

be costly, while path search in a PRM can take a prohibitively long time to get a real-time response from the robot.

B. CONTRIBUTION

We propose MARS, a **M**ulti-p**A**th **R**eplanning **S**trategy designed to quickly provide a suitable solution in complex and high-dimensional search spaces. MARS core consists of building a graph that connects paths to the same goal, allowing the exploitation of a set of paths pre-calculated. When an obstacle blocks the current path, it looks for a new one by trying to connect the current path to the other available ones. If a connection is found, it can search for better solutions during the remaining time.

Exploiting a set of pre-computed paths to find a solution is partially new in the literature. Most algorithms, indeed, involve a tree pruning phase and its subsequent reconstruction (DRRT [16], Anytime DRRT [17], RRF [32], and Multi-partite RRT [19]). The few state-of-the-art approaches that exploit a population of trajectories for replanning use evolutionary computation [36], or Gaussian processes and factor graphs [37], [38]. Instead, our multi-path strategy connects the current path before the obstacle to a node closer than the goal and then uses the available sub-path from that node to the goal. Informed sampling allows to improve the plan faster over time.

The paper presents contributions beyond the method's novelty. First, the strategy is not tied to a specific sampling-based path-planning algorithm. It combines informed sampling, directed graphs, problem knowledge reuse, and lazy collision checking to reduce the computational load. Informed sampling is used to focus the search for better solutions, and the directed graph allows the results obtained in previous iterations to be exploited. Second, it introduces the *first-order connections* and *second-order connections* to handle multiple paths to the goal in the same tree. Third, it presents a multithread architecture for executing the trajectory with continuous replanning and collision checking. This architecture can be used with various replanning algorithms and reduces the computational load on the replanner.

This work represents a revision and extension of previous papers [39], [40]. There is now a great deal of reuse of the tree grown at each iteration to find solutions faster. A directed graph is introduced, which extends the tree data structure, allows intermediate connections between paths to be handled more efficiently, and allows multiple existing solutions to be searched toward the goal. The replanning architecture and its role are explained in more detail. The open-source C++ code for the proposed replanner and architecture is at [41].

II. METHOD

Denote X, $X_{\text{obs}} \subset X$ as the space of all configurations and all configurations in a collision, respectively. Given an initial configuration $x_{\text{start}} \in X_{\text{free}} \equiv \text{closure}(X \setminus X_{\text{obs}})$ and a goal configuration $x_{\text{goal}} \in X_{\text{free}}$, a path planning problem solver finds a curve $\sigma : [0, 1] \to X_{\text{free}}$ such that $\sigma(0) = x_{\text{start}}$ and $\sigma(1) = x_{\text{goal}}$. A solution curve to such a problem is a *feasible path*. Furthermore, an *optimal path* is a feasible path σ^* such that:

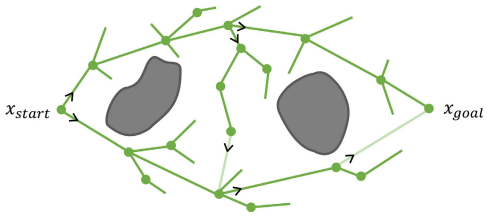


FIGURE 1. The graph *G* and tree *T*. *T* is represented with green connections; *G* extends *T* with light green connections. Arrows indicate the direction of travel of the directed graph.

$$\sigma^* = \underset{\sigma \in \Sigma_{\text{free}}}{\text{argmin}}\, c(\sigma), \tag{1}$$

where $c : \Sigma_{\text{free}} \to \mathbb{R}$ is a cost function that associates a cost with any feasible path $\sigma \in \Sigma_{\text{free}}$. Often, the cost function c is the length of the path, $\|\sigma\|$, so that the optimal motion plan is the shortest collision-free path from x_{start} to x_{goal}. Most path planners consider X_{obs} constant over time, with limitations in unstructured environments. A reactive behavior modifies the robot's motion at run-time to prevent collisions and accomplish the intended objective and is implemented via *online path replanning*.

We consider two types of data structures embedded in X: a graph $G := (V, E)$ and a tree $T := (V_T, E_T)$ (Figure 1). V and E are the sets of nodes and connections (or edges) of G, respectively, while $V_T \subset V$ and $E_T \subset E$ are the sets of nodes and connections of T. Every node of T possesses a single parent and can have multiple children except the tree's root, which has no parent. Denote as:

***first-order connections*:** connections between nodes in T, *i.e.,* E_T. Once the tree is connected to the goal node, the path is obtained following the parents from the goal.

***second-order connections*:** connections to nodes that already have a parent in the tree and, therefore, an incoming first-order connection. The set of *second-order connections* is $\text{cl}(E \setminus E_T)$. G, which extends T, knows about these connections, but T doesn't.

This formulation handles multiple paths and interconnecting portions of the same tree. Both T and G will be exploited in this paper: T is used to build multiple new solutions, and G is queried to extract the best one.

A. REPLANNING ARCHITECTURE

This architecture handles trajectory execution, continuous collision checking, and replanning. It is not bound to a specific replanning algorithm. It offers the advantage of relieving the replanner of the task of collision checking along the path followed, thus reducing its computational load. Our replanning algorithm exploits this option (Sec. II-B).

The architecture relies on three parallel threads (Alg. 1):

Trajectory Execution Thread micro-interpolates the robot's current trajectory τ to send the new command to the robot controller at a high rate.

TABLE 1. Notation.

$\sigma_i : [0,1] \to X_{\text{free}}$	is a path from x_{start} to x_{goal};
σ_{curr}	is the current robot path;
S	is the set of other available paths calculated at the beginning, thus excluding the current robot path σ_{curr};
$w_i = (x_1, \ldots, x_M)$	is the sequence of waypoints (nodes) that composes σ_i, where $x_1 = x_{\text{start}}$ and $x_M = x_{\text{goal}}$;
$\sigma_i[x_j, x_k]$	is the portion of σ_i from $x_j \in \sigma_i$ to $x_k \in \sigma_i$;
$c : \Sigma \to \mathbb{R}_{\geq 0}$	is a cost function that associates a positive real cost to a feasible path and $+\infty$ if the path is infeasible;
$\|\cdot\|$	is the Euclidean norm operator;
$\|\sigma\| : \Sigma \to \mathbb{R}_{\geq 0}$	is the length of a path $\sigma \in \Sigma$;
$\sigma_i \cup \sigma_j : \Sigma \to \Sigma$	is a function that concatenates σ_j with σ_i (being $\sigma_i(1) = \sigma_j(0)$).
$G := (V, E)$	is a directed graph and V, E the set of nodes and connections, respectively.
$T := (V_T, E_T)$	is the tree associated with path σ, with $V_T \subset V$ and $E_T \subset E$.
τ	is a trajectory associated with path σ respecting the kinodynamic constraints of the robot.
x_{curr}	is the current robot node, *i.e.*, the next node of σ_{curr} closest to the robot position.
$x_{\text{before}}, x_{\text{after}} \in w_{\text{curr}}$	are respectively the last node before the obstacle and the first node after.

Algorithm 1 The Threads That Operate in Parallel. They Requires σ_{curr} and S as Common Input

```
1: Thread Trajectory execution
2:     t = t + Δt
3:     state ← sampleTrajectory(τ, t)
4:     sendToController(state)
5: Thread Collision check
6:     σ_subpath ← σ_curr[x_curr, x_goal]
7:     P ← S ∪ σ_subpath;
8:     for σ_j ∈ P do
9:         free, x_before, x_after ←
               checkCollision(σ_j)
10:        if free then
11:            c_σj ← c(σ_j)
12:        else
13:            c_σj ← +∞
14: Thread Path replanning
15:    x_curr ← projectOnPath(state, σ_curr)
16:    σ_RP, success ←
           replan(σ_curr, x_curr, S, max_time)
17:    if success then
18:        σ_curr ← σ_RP
19:        τ ← computeTrajectory(σ_curr)
20:    else if dist from obstacle ≤ min_dist then
21:        sendRobotStop()
```

Algorithm 2 MARS: High-Level Description

```
1: Define P as the set of available paths
2: Define Q_1 as the queue of the nodes of σ_curr between x_curr
   and the obstacle
3: Sort Q_1 by some criterion
4: for each node x_n ∈ Q_1 do
5:     Insert the nodes of each σ ∈ P into a queue Q_2
6:     Sort Q_2 by some criterion
7:     for each x_j ∈ Q_2 do
8:         Define the informed set on the best cost up to now
9:         Get the subtree rooted in x_n from the informed set
10:        Grow the subtree in the informed set to reach x_j
11:    Update Q_1 if a solution was found
12: Get the best path from the graph
```

Collision Checking Thread checks for the collisions along the path in execution. It considers σ_{subpath}, which is the part of the current path σ_{curr} from the current robot configuration x_{curr} to the goal. This thread also checks for paths $\sigma_i \in S$ if MARS is running. These checks are parallelizable. This thread computes which is the last node of the path before the obstacle ($x_{\text{before}} \in w_{\text{curr}}$) and the first one after ($x_{\text{after}} \in w_{\text{curr}}$) (Fig. 2a).

Path Replanning Thread invokes the replanning algorithm to find a path when the current one is obstructed or to optimize the current solution. Before that, a routine projects the robot state on σ_{curr} to obtain x_{curr}. If the replanner finds a new path, it assigns it to σ_{curr}. The replanner gets the maximum computation time as an input, which may change depending on the situation, *e.g.,* it can be shorter when the path is obstructed and longer otherwise. If the replanner fails and the robot's distance from the obstacle is critical, a robot stop signal is triggered, and a contingency plan must be implemented.

This architecture allows the replanning algorithm to avoid collision checking of the current path (and other available paths $\sigma_i \in S$). In this way, the replanner has more time to search for a solution or to improve the current one.

B. REPLANNING ALGORITHM AT A GLANCE

The section introduces MARS, a path replanner that exploits a set of pre-computed paths to find a new solution quickly, even in high-dimensional and complex scenarios. MARS computes a free path when the current one becomes infeasible and optimizes it during the execution. MARS follows an anytime approach so that it gets the first solution quickly and then tries to improve it over time.

Without loss of generality, we will consider that the cost of a path is represented by its length:

$$c(\sigma) = \|\sigma\| = \sum_{i=1}^{M-1} \|x_{i+1} - x_i\| \tag{2}$$

where $(x_1, x_2, .., x_M)$ are the nodes of path σ. This cost function is commonly used, but the method can be easily extended to other cost functions as well, provided that an admissible informed set is available.

Consider Algorithm 2, where a high-level pseudo-code is reported, while the details of the implementations are in Section II-D. MARS uses a set of pre-computed paths to find a new solution. The first step consists of adding the valid part of the current path to the set of available paths S. Then, all the nodes of the current path between the robot and the obstacle are inserted into a queue Q_1. Some criterion sorts this queue (*e.g.,* the closest nodes to the robot are the first processed). The algorithm tries to connect each node $x_n \in Q_1$ to the nodes of the other available paths, inserted into a queue Q_2, and sorted by some criterion (*e.g.,* the distance from the node x_n). The next step consists of finding a path connecting $x_n \in Q_1$ to $x_j \in Q_2$. The algorithm first defines an admissible set with the cost of the best solution found up to now and then selects the subtree with root in x_n contained in this set. It then invokes a sampling-based path planner to grow the subtree until it reaches x_j (Fig. 2a). Since x_j has a parent node in the tree, a *second-order* connection is used to connect the subtree to x_j. Second-order connections are not visible on the tree but on the directed graph. MARS creates multiple solutions from the robot's configuration to the goal, and then it extracts the best one from the graph (Fig. 2b).

C. PROPERTIES OF THE REPLANNING ALGORITHM

Sampling-based path planners are probabilistically complete if the probability of finding a solution is equal to one with infinite samples. The planners are almost-surely asymptotically optimal if the solution cost converges to the optimal one as the number of samples goes to infinity. These properties are not provable for replanners without any assumptions on obstacle dynamics. It is easy to build counterexamples in which an obstacle obstructs any new given solution, preventing the robot from finding a path to the goal. Nonetheless, assuming a static scene from a specific instant of time onward, we can prove these properties for MARS.

1) PROBABILISTIC COMPLETENESS

MARS searches for a path from a generic start node $x_n \in Q_1$ to a generic goal node $x_j \in Q_2$, which do not necessarily correspond to the current robot configuration and the goal. Note that the subpaths from the robot configuration to x_n and from x_j to the goal are always feasible. The problem, therefore, reduces to proving that the algorithm is complete in searching for a path from x_n to x_j. A sufficient condition for an RRT-like planner to be probabilistically complete is to draw samples with a probability greater than zero across the search space. When the current solution is obstructed, we superimpose that c_i equals infinity (see (7) in Sec. II-D). The ellipsoid, therefore, comprises the entire search space sampled with non-zero probability. With an infinite number of samples, the algorithm can find a feasible solution if one exists.

2) ASYMPTOTIC OPTIMALITY

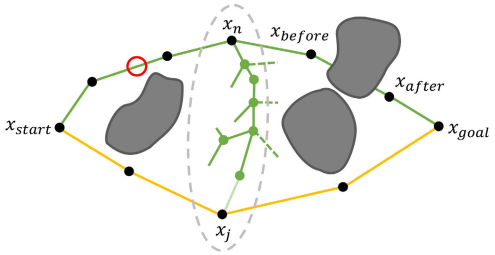
MARS is asymptotic optimal when it uses an asymptotically optimal planner, such as RRT*. When MARS tries to optimize the current path, it chooses pairs of nodes (x_n, x_j) such that $x_n \in Q_1$ and $x_j \in Q_2$. When x_n corresponds to the current robot configuration and x_j corresponds to the goal, the algorithm behaves like Informed RRT* [30] and is therefore asymptotically optimal. For all other pairs of nodes, MARS optimizes the connecting path between the two.

Algorithm 4 MARS

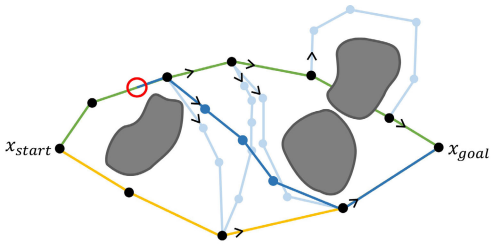
Input: $\sigma_{\text{curr}}, x_{\text{curr}}, S = \{\sigma_1, \ldots, \sigma_N\}, G$, max_time
Ensure: replanned path σ_{RP}
mergeTrees($\sigma_{\text{curr}}, S, G$)
$\sigma_{\text{RP}} \leftarrow \sigma_{\text{curr}}[x_{\text{curr}}, x_{\text{goal}}]$; $c_{\text{RP}} \leftarrow c(\sigma_{\text{RP}})$
$P \leftarrow S$
if $c_{\text{RP}} = \infty$ **then**
  $P \xleftarrow{+} \sigma_{\text{RP}}[x_{\text{after}}, x_{\text{goal}}]$
  $Q_1 \leftarrow w_{\text{RP}}[x_{\text{curr}}, x_{\text{before}}]$
else
  $P \xleftarrow{+} \sigma_{\text{RP}}$
  $Q_1 \leftarrow w_{\text{RP}}$
Q_1.sortQueue()
while ¬ isEmpty(Q_1) & $t <$ max_time **do**
  $x_n \leftarrow Q_1$.pop()
  $t_{\text{MAX}} \leftarrow$ max_time $- t$
  $\sigma_{\text{c2p}} \leftarrow$ connectToPaths($x_n, \sigma_{\text{RP}}, P, t_{\text{MAX}}$)
  if ¬ isEmpty(σ_{c2p}) **then**
    $\sigma_{\text{cn}} \leftarrow \sigma_{\text{RP}}[x_{\text{curr}}, x_n]$
    $\sigma_{\text{cand}} \leftarrow \sigma_{\text{cn}} \cup \sigma_{\text{c2p}}$
    if $c(\sigma_{\text{cand}}) < c_{\text{RP}}$ **then**
      $\sigma_{\text{RP}} \leftarrow \sigma_{\text{cand}}$; $c_{\text{RP}} \leftarrow c(\sigma_{\text{cand}})$
      sol_found $\leftarrow$ True
      Q_1.updateQueue(w_{RP})
if sol_found **then**
  $t_{\text{MAX}} \leftarrow$ max_time $- t$
  ($\sigma_{\text{RP}}, c_{\text{RP}}$) $\leftarrow$ searchBetterPath($x_{\text{curr}}, x_{\text{goal}}, G, \sigma_{\text{RP}}, t_{\text{MAX}}$)

Algorithm 4a Merge the Trees

procedure mergeTrees($\sigma_{\text{curr}}, S = \{\sigma_1, \ldots, \sigma_N\}, G$)
  $T \leftarrow \sigma_{\text{curr}}$.tree()
  for σ_j in S **do**
    $T_j \leftarrow \sigma_j$.tree()
    if $T_j \neq T$ **then**
      $T \leftarrow T \cup T_j$
  for σ_j in S **do**
    σ_j.setTree(T)
  if $T \not\subset G$ **then**
    G.add(T)
  return updated tree T and directed graph G



(a) The grey-dotted ellipse is the informed set. The green-dotted branches are outside the ellipse, so they are not considered in the subtree. The second-order connection to x_j is the light green line, which is not visible on the tree but is visible on the graph.



(b) Simplified representation of the result of MARS. The light blue paths connect the current path to the available paths found during the algorithm's execution. The blue path is the best solution extracted from the graph.

FIGURE 2. Representation of the subtree rooted in x_n and built to reach x_j. The green path is the current path; the yellow path is another available path. The red circle represents the current robot configuration x_{curr}.

D. DETAILED DESCRIPTION OF THE REPLANNING ALGORITHM

Algorithm 4 is the meta-code of MARS. First, MARS merges the current path tree with those of the available paths (Alg. 4a). MARS connects paths that may have been computed with independent trees and builds a graph to extract the best solution. The trees of the available paths need to be merged with that of the current path. Since all trees have the same root (paths start at the same node), this procedure is trivial. Ultimately, all paths share the same tree, which belongs to graph G.

The solution path σ_{RP} is initialized with the subpath from x_{curr} to the goal. The set of available paths S is enriched with the valid portion of the subpath σ_{RP} to form the set P.

All the nodes of $\sigma_{\text{RP}}[x_{\text{curr}}, x_{\text{before}}]$ are inserted into the queue Q_1. Then, the algorithm computes multiple paths connecting $x_n \in Q_1$ to the other available paths using connectingToPaths and assigns the best solution to σ_{c2p}; now, the algorithm builds the candidate solution concatenating the subpath from x_{curr} to x_n with σ_{c2p}. If the candidate solution has a lower cost than σ_{RP}, it becomes σ_{RP}. At this point, Q_1 is updated (*e.g.*, Q_1 is emptied, and the nodes of the new solution not previously used are inserted into the queue). Note that the number of nodes in Q_1 is small, so the sorting function is not computationally demanding. In the end, the graph G is searched for a path better than σ_{RP}.

The core of the replanner is connectToPaths (Alg. 4b), called on each $x_n \in Q_1$. This function takes as input $x_n \in Q_1$ and tries to find paths connecting it to the nodes of the available paths P. As output, it returns the best path σ_{sol} from x_n to x_{goal} found so far. To do this, first, it populates the queue Q_2 with the nodes of each path $\sigma_j \in P$ and sorts it based on some criteria (*e.g.*, on the distance from x_n). Then, a path from x_n to each $x_j \in Q_2$ is computed by informedPlan; the path σ_{conn} found is then concatenated with the subpath σ_{jg} from x_j to x_{goal} if it results in a better solution than the best one found so far.

The graph search (Alg. 4c) takes a lazy approach to collision checking. The algorithm computes a map of paths sorted by cost; then, it scrolls through the paths contained in the map until it finds a valid one. During this validation phase, only connections that have not been checked during this replanning call are considered: connections that make up the available paths, those of the connecting paths found, and others previously checked are not re-checked. The map is computed using a depth-first search algorithm starting from the goal node. Specifically, let be x_g the goal node, x_s the start node, x_i the node considered at the *i-th* step of the search, and c_{gi} the cost of the branch from x_g to x_i followed in this iteration. The search along this branch stops when

$$c_{gi} + \|x_i - x_s\| > c_{\text{better}} \tag{3}$$

Both first-order and second-order connections are considered during the search in the graph G.

The most demanding part of Algorithm 4 is Sub-Algorithm 4b, which must be very efficient.

First, we speed up the search by purging the nodes $x_j \in Q_2$ that do not improve the solution. Let be $x_n \in Q_1$ and $x_j \in Q_2$ the the root and goal nodes of the connecting path σ_{conn} (Fig. 3c). The candidate solution $\sigma_{sol} = \sigma_{\text{conn}} \cup \sigma_j[x_j, x_{\text{goal}}]$ is better than the current best solution $\sigma_i[x_n, x_{\text{goal}}]$, if

$$c(\sigma_{\text{conn}}) < c(\sigma_i[x_n, x_{\text{goal}}]) - c(\sigma_j[x_j, x_{\text{goal}}]) \tag{4}$$

The lower bound of $c(\sigma_{\text{conn}})$ is the Euclidean distance from x_n to x_j. So, x_j improves the current path if

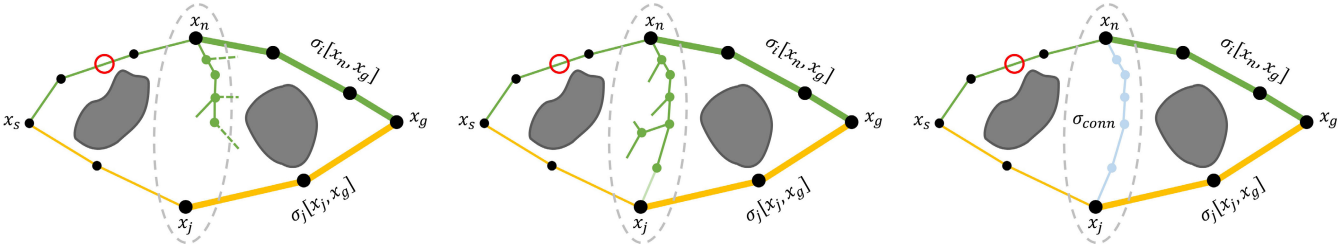
$$\|x_n - x_j\| < c(\sigma_i[x_n, x_{\text{goal}}]) - c(\sigma_j[x_j, x_{\text{goal}}]) \tag{5}$$

If (5) does not hold, x_j is discarded by connectToPaths. Note that, if $\sigma_i[x_n, x_{\text{goal}}]$ is infeasible, (4) and (5) always hold as the path has infinite cost. When a solution is found, (5) is updated with the new path's cost $c(\sigma_{\text{sol}})$, such that

$$\|x_n - x_j\| < c(\sigma_{\text{sol}}) - c(\sigma_j[x_j, x_{\text{goal}}]) \tag{6}$$

In this way, only nodes with a greater than zero probability of improving the current solution are considered.

To enhance the speed of Sub-Algorithm 4b, we also exploit the *informed sampling* [4] in informedPlan. When searching

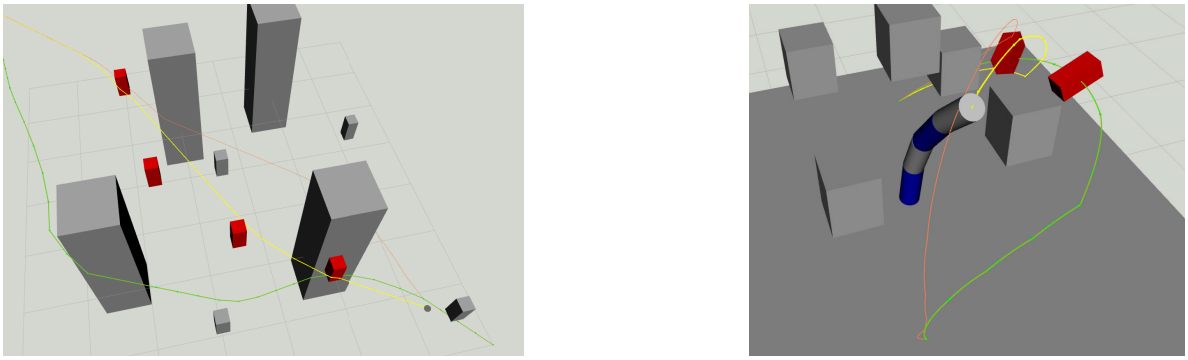


(a) The subtree rooted in x_n belonging to the ellipsoid E_{ll} (7). The dotted green line are not considered because not in E_{ll}.

(b) The subtree is grown to reach x_j. The last connection is a second-order connection.

(c) The connecting path from x_n to x_j found (light blue path).

FIGURE 3. Subtree (green) and subpaths (bold lines) considered to reduce the computational load of connectToPaths. x_s and x_g are used in place of x_{start} and x_{goal} for brevity.



(a) Medium 3dof scenario.

(b) 6dof scenario.

FIGURE 4. Screenshots from the tests of MARS. The yellow line is the robot's current path, while the other colored lines are the other available paths. Grey boxes are the fixed obstacles, and red boxes are the mobile ones.

for a path connecting x_n to x_j, the search space can be shrunk to the hyper-ellipsoid

$$E_{ll} = \{x \in X_{\text{free}} \mid \|x - x_n\| + \|x_j - x\| < c_i\} \qquad (7)$$

where $c_i = c(\sigma_{\text{sol}}) - c(\sigma_j[x_j, x_{\text{goal}}])$. Equation (7) represents an admissible set, so the nodes outside the ellipsoid are discarded since they cannot improve the solution (Fig. 3). informedPlan checks if an existing path between x_n and x_j with a cost lower than $c_{\max}$ exists. This search is in G_{T_s}, *i.e.*, the subgraph that contains the *first-order* and the *second-order connections* between nodes of T_s or to x_j. A ready-to-use solution can exist thanks to previous replanning iterations. Otherwise, growInEllipsoid invokes growTree[1] that grows the subtree in the ellipsoid E_{ll} to reach x_j. If a solution is found, replacing the connection to x_j with a *second-order connection* is the only caveat. growInEllipsoid adopts a lazy collision check. The connections already in the subtree are checked only when the path is found. The branches with a collision are hidden, and growth begins again.

Figure 3 reports one iteration of connectToPaths. We select $x_j \in Q_2$ as a valid node to try to connect to by exploiting $\sigma_i[x_n, x_{\text{goal}}]$ and $\sigma_j[x_j, x_{\text{goal}}]$ (the bold green and yellow lines). The ellipsoid E_{ll} (grey dotted ellipse) is defined by (7), and the already existing subtree rooted in x_n and contained in E_{ll} is selected (Fig. 3a). The connections of the subtree outside E_{ll} are not considered (dotted green lines). Then, the subtree is grown in E_{ll} using a path planner (Fig. 3b). Once x_j is reached, a second-order connection is created between it and the subtree (light green line). The light blue path (Fig. 3c) is the σ_{conn} found. If the cost of $\sigma_{\text{conn}} \cup \sigma_j[x_j, x_{\text{goal}}]$ is less than the cost of $\sigma_i[x_n, x_{\text{goal}}]$, it becomes the new candidate solution. Then, the procedure is repeated for each $x_j \in Q_2$ that satisfies (6). Once the whole Q_2 has been considered, Q_1 is updated, and a new x_n popped.

[1] growTree is a generic sampling-based planner, and the approach is easily extendable to bi-directional ones.

TABLE 2. Scenarios data used during the experiments.

n.	Name	State size	Fixed obstacles	Moving obstacles
#1	Small 3dof	$3\text{m} \times 3\text{m} \times 3\text{m}$	3	3
#2	Medium 3dof	$7\text{m} \times 7\text{m} \times 3\text{m}$	8	6
#3	Large 3dof	$12\text{m} \times 12\text{m} \times 3\text{m}$	10	10
#4	6dof	$[-\frac{\pi}{2}, \frac{\pi}{2}]$ rad	6	3
#5	12dof	$[-\frac{\pi}{2}, \frac{\pi}{2}]$ rad	6	3
#6	18dof	$[-\frac{\pi}{2}, \frac{\pi}{2}]$ rad	6	3

III. EXPERIMENTAL RESULTS

A. NUMERICAL SIMULATIONS

A simulation testing was conducted to compare MARS with some state-of-the-art path replanners. In particular, MARS was compared with DRRT, Anytime DRRT, MPRRT, and with [31], which we will refer to by the acronym DRRT*

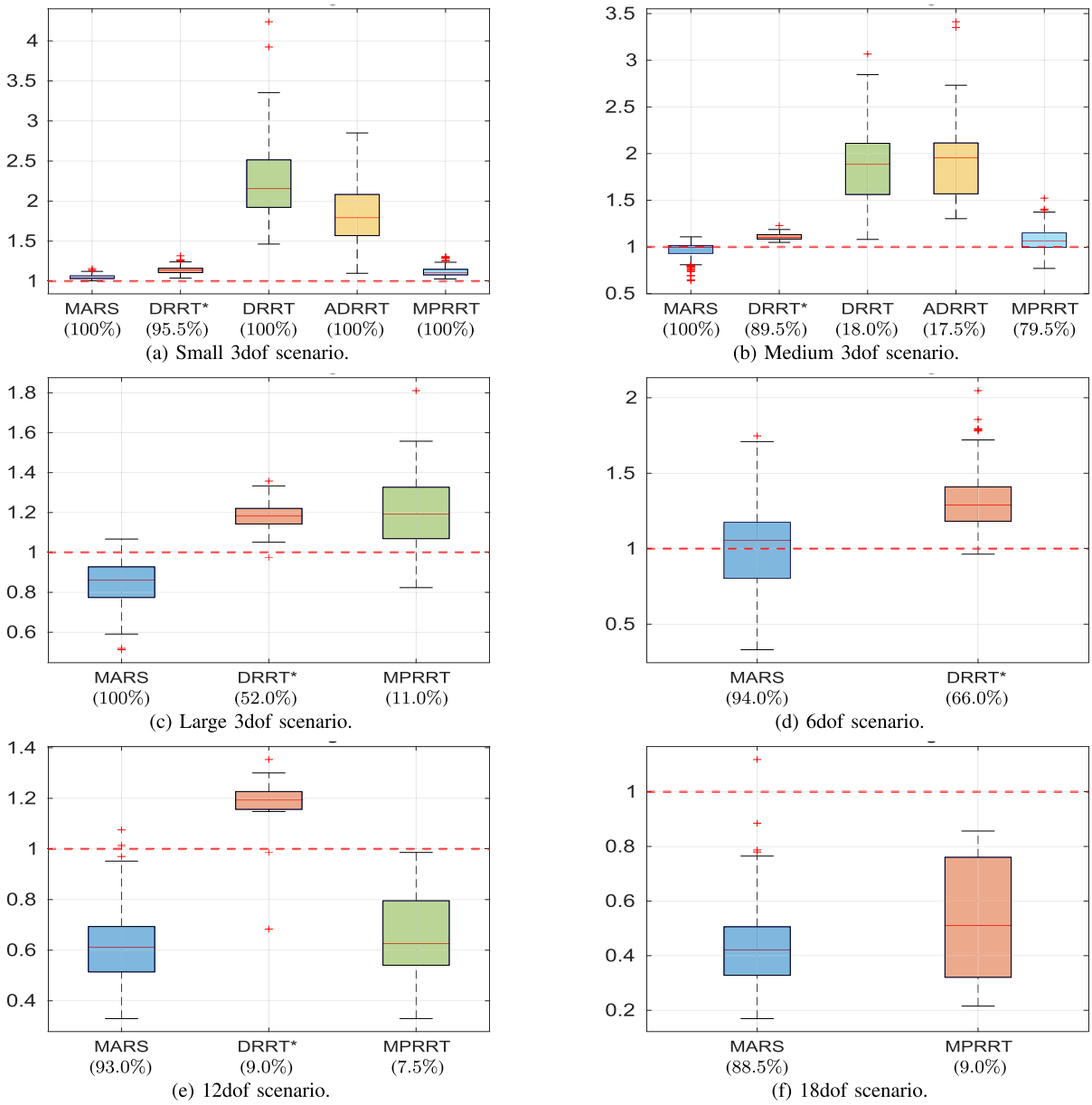


FIGURE 5. **Normalized path length in each scenario. When a boxplot is below the red dashed line, the replanner produces a shorter path than the initial one. The success rate is listed under each replanner's name. Only replanners with a success rate greater than 5% are shown. Maximum replanning time 200ms.**

for convenience. The tests were conducted using ROS and *MoveIt!* on a computer with a 2.80 GHz CPU. The code implemented can be found here [41]. Each of the algorithms has been integrated into the architecture of Algorithm 1.

They were tested in 6 scenarios, which differed in the size of the search space (*i.e.*, degrees of freedom of the robot):

Scenarios 1, 2, and 3 consist of a point robot moving in a 3-dimensional space. There are fixed obstacles of various sizes in each of the scenarios (Table 2);

Scenarios 4, 5 and 6 provide a 6-, 12-, and 18-DoF anthropomorphic robot, respectively. Each of these scenarios shares the same fixed obstacles.

The test consists of 200 executions for each scenario, divided into 20 queries and 10 iterations per query. At each query corresponds a different pair of start and goal configurations. For each start-goal pair, the test is repeated 10 times. Initially, the robot path is calculated. In the case of MARS, the set of the other paths (two paths) is also computed at the beginning. The planner used is RRT-Connect [42], followed by an optimization procedure [43]. During execution, new obstacles randomly appear, obstructing the robot's path and forcing the replanner to find a new path. Table 2 shows the number of fixed and moving obstacles. Each algorithm is assigned a maximum replan time of 200ms. The frequency of the *trajectory execution thread* and the *collision checking*

Algorithm 4b Connect Node to Paths

procedure conntectToPaths(x_n, σ_i, P, max_time)

  function growInEllipsoid(T, E_{ll}, x_j, t_{MAX})
    while $\neg$ *ok* & *iter* $<$ max_iter & $t < t_{\text{MAX}}$ **do**
      $iter = iter + 1$
      $q \leftarrow$ sampleInEllipsoid(E_{ll})
      $x_{\text{new}} \leftarrow T$.growTree($q$)
      if $||x_{\text{new}} - x_j|| <$ min_dist **then**
        if path from tree root to x_j is valid **then**
          create a second-order connection from x_{new} to x_j
          $ok \leftarrow$ True
        else
          hide the invalid branch temporarily from T
    return *ok*

  function informedPlan(x_n, x_j, $c_{\max}$, max_time)
    $E_{ll} \leftarrow$ ellipsoid defined by $c_{\max}$
    $T_s \leftarrow$ subtree rooted in x_n contained in E_{ll}
    $G_{T_s} \leftarrow$ the subgraph of G containing T_s
    $(\sigma_{\text{nj}}, c_{\text{nj}}) \leftarrow$ searchBetterPath(x_n, x_j, G_{T_s}, $c_{\max}$, max_time)
    if $\neg$ isEmpty(σ_{nj}) **then**
      $ok \leftarrow$ True
    else
      $t_{\text{MAX}} \leftarrow$ max_time $- t$
      $ok \leftarrow$ growInEllipsoid(T_s, E_{ll}, x_j, t_{MAX})
      if *ok* **then**
        $(\sigma_{\text{nj}}, c_{\text{nj}}) \leftarrow$ searchBetterPath(x_n, x_j, G_{T_s}, $c_{\max}$)
    return σ_{nj} from x_n to x_j, c_{nj}, *ok*

▷ Main connectToPaths Code

  $\sigma_{\text{sol}} \leftarrow \sigma_i[x_n, x_{\text{goal}}]$; $c_{\text{sol}} \leftarrow c(\sigma_{\text{sol}})$
  for $\sigma_j \in P$, $x_j \in w_j$ **do**
    Q_2.addTuple(x_j, σ_j)
  Q_2.sortQueue()
  while $\neg$ isEmpty(Q_2) & $t <$ max_time **do**
    $(x_j, \sigma_j) \leftarrow Q_2$.pop()
    $\sigma_{\text{jg}} \leftarrow \sigma_j[x_j, x_{\text{goal}}]$; $c_{\text{jg}} \leftarrow c(\sigma_{\text{jg}})$
    $c_{\max} \leftarrow c_{\text{sol}} - c_{\text{jg}}$
    if $\|x_n - x_j\| < c_{\max}$ **then**
      $t_{\text{MAX}} \leftarrow$ max_time $- t$
      $\sigma_{\text{conn}}, c_{\text{conn}}, ok \leftarrow$ informedPlan(x_n, x_j, $c_{\max}$, t_{MAX})
      if *ok* **then**
        $c_{\text{new}} \leftarrow c_{\text{conn}} + c_{\text{jg}}$
        if $c_{\text{new}} < c_{\text{sol}}$ **then**
          $\sigma_{\text{sol}} \leftarrow \sigma_{\text{conn}} \cup \sigma_{\text{jg}}$; $c_{\text{sol}} \leftarrow c_{\text{new}}$
    Q_2.removeTuple(x_j, σ_j)
  return a new path σ_{sol} from x_n to x_{goal}

Algorithm 4c Search a Better Path in the Graph

procedure *searchBetterPath*(x_s, x_g, G, σ_c OR $c_{\max}$, t_{MAX})
  if $\neg$ isEmpty(σ_c) **then**
    $\sigma_{\text{better}} \leftarrow \sigma_c$; $c_{\text{better}} \leftarrow c(\sigma_c)$
  else
    $\sigma_{\text{better}} \leftarrow$ VOID; $c_{\text{better}} \leftarrow c_{\max}$
  if t_{MAX} is VOID **then**
    $t_{\text{MAX}} \leftarrow \infty$
  sorted_map $\leftarrow G$.lowerCostPaths(x_s, x_g, c_{better}, t_{MAX})
  while $\neg$ *ok* & $\neg$ isEmpty(*sorted_map*) **do**
    $(\sigma_{\text{new}}, c_{\text{new}}) \leftarrow$ *sorted_map*.pop()
    if σ_{new}.valid() **then**
      $\sigma_{\text{better}} \leftarrow \sigma_{\text{new}}$; $c_{\text{better}} \leftarrow c_{\text{new}}$
      ok $\leftarrow$ True
  return σ_{better} from x_s to x_g, c_{better}

thread are 500Hz, and 30Hz. The queue Q_1 of MARS is sorted on the distance along the path from the current robot configuration. The queue Q_2 is sorted on the distance from the processed node $x_n \in Q_1$.

The algorithms are evaluated using the following metrics:

Success rate: denote N_{succ} and N_{tests} as the number of tests executed without collisions and the number of tests performed in each scenario, respectively. The success rate

$$S\% = N_{\text{succ}}/N_{\text{tests}}$$

is the percentage of tests in which the robot reached the goal without collisions.

Collision rate: denote $N_{\text{collisions}}$, N_{tests} and N_{obs} as the number of obstacles the robot collided with, of tests performed and of random obstacles in the considered scenario, respectively. The collision rate

$$C\% = N_{\text{collisions}}/((1 - S\%)\, N_{\text{tests}}\, N_{obs})$$

indicates the number of obstacles the robot collided with when the replanner failed. The denominator is the number of obstacles during unsuccessful iterations. For robots with a built-in safety stop procedure that avoid collisions, this metric can be used as a proxy for the number of safety stops.

Normalized path length: denote $\|\sigma_{\text{real}}\|$ and $\|\sigma_{\text{init}}\|$ as the length of the path traversed by the robot and the length of the path computed at the beginning of the iteration, respectively. The

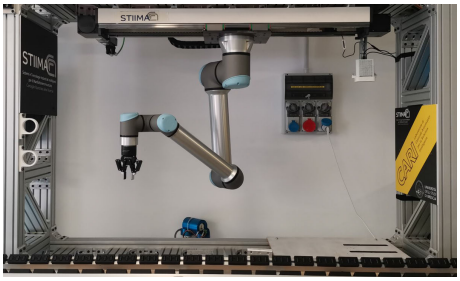
$$n.p.l. = \|\sigma_{\text{real}}\|/\|\sigma_{\text{init}}\|$$

indicates how longer (due to obstacle avoidance) or shorter (due to path optimization) the path is compared to the initial one. Since σ_{real} depends on σ_{init}, its length is normalized to obtain a comparable measure across tests.

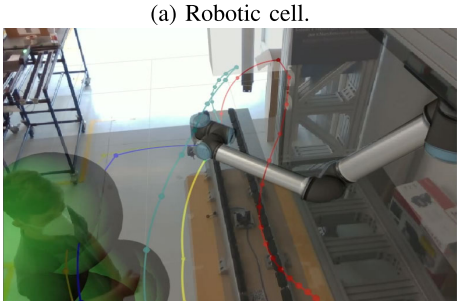
Figure 4 shows screenshots of two tests of MARS in a 3 DoF and 6 DoF scenario. Table 3 shows the success rate $S\%$ and collision rate $C\%$ of the algorithms in each scenario. Figure 5 shows the boxplot of the normalized path length $n.p.l$ for replanners that achieved at least a 5% success rate. All planners have a high success rate for scenario 1. As the complexity grows, MARS maintains a high success rate ($S\% \geq 88.5\%$). In scenarios 5 and 6, MPRRT and DRRT* achieve a success rate of 9.0% and 4.0%, respectively, whereas DRRT and Anytime DRRT always fail. MPRRT performs better than DRRT and Anytime DRRT in scenario 6 because it plans on multiple parallel threads, and its collision rate remains lower than DRRT and Anytime DRRT.

TABLE 3. Success rate and Collision rate in different scenarios. Maximum replanning time 200ms.

	Small 3dof		Medium 3dof		Large 3dof		6dof		12dof		18dof	
	S%	C%	S%	C%	S%	C%	S%	C%	S%	C%	S%	C%
MARS	**100.0**	**0.0**	**100.0**	**0.0**	**100.0**	**0.0**	**94.0**	**38.9**	**93.0**	**36.6**	**88.5**	**37.3**
DRRT*	95.5	33.3	89.5	19.0	52.0	34.8	66.0	44.8	9.0	77.1	4.0	82.2
DRRT	100.0	0.0	18.0	55.2	1.0	79.3	0.0	98.9	0.0	99.7	0.0	99.8
Anytime DRRT	100.0	0.0	17.5	56.3	1.50	80.2	0.0	99.4	0.0	99.3	0.0	99.7
MPRRT	100.0	0.0	79.5	23.2	11.0	29.3	2.0	92.7	7.5	76.4	9.0	72.2



(a) Robotic cell.



(b) Test on the real cell.

FIGURE 6. Collaborative robotic cell with UR10e mounted upside down and Intel RealSense D435.

Figure 5 shows the *n.p.l* for the algorithms with a $S\% \geq 5\%$. When a boxplot is below the red dashed line, the replanner produces a shorter path than the one computed initially. Anytime DRRT, MPRRT, and MARS continuously try to improve the solution, even when it is unobstructed. DRRT and DRRT* replan only when the path is obstructed. DRRT ends as soon as it finds a feasible solution. DRRT* uses all remaining replanning time to improve the found solution.

In scenarios 1-2 MARS, DDRT* and MPRRT provide comparable *n.p.l*. (Figure 5a and 5b). The performance difference compared to MARS increases with the complexity of the problem. MARS outperforms the baselines in all those situations where the time to compute and optimize the initial path is critical; the algorithm, in these cases, will act as an online optimizer. Anytime DRRT seems equivalent to DRRT, probably because most of the replanning time is spent by DRRT on finding a feasible solution, and the remaining time devoted to its improvement is small.

In scenarios 5 and 6, MARS and MPRRT have comparable *n.p.l*. results. However, they are dramatically different in terms of success rate (93% and 88.5% of MARS compared to 7.5% and 9% of MPRRT).

B. REAL-WORLD EXPERIMENTS

MARS and the replanning architecture were tested in the collaborative robotic cell of Figure 6. Specifically, the setup consists of a 6-DoF UR10e manipulator mounted upside down and an Intel Realsense D435 to track the operator's position. A speed scaling module was active during the experiments to slow down the trajectory depending on the human-robot distance for safety reasons (see Section III.F of [44], and [45] for more details about the scaling module). In the experiments, the robot moved from a start position to an end position. At the same time, a human obstructed its initially calculated path, forcing the algorithm to search for a new valid path and to optimize it over time. A demonstration video of the application is attached to this paper.

IV. CONCLUSIONS AND FUTURE WORKS

This paper proposed MARS, a sampling-based path replanning algorithm for complex, high-dimensional scenarios. The novelty of the algorithm is the exploitation of a set of pre-computed paths to find and optimize over time a new path when the current one is obstructed by an unexpected obstacle. The algorithm determines which nodes of the available paths to connect to based on the cost of the current solution and gradually builds a directed graph to exploit the results of previous iterations. Informed sampling, subtrees reuse, and lazy collision checking allows for fewer calculations and finding better solutions more quickly. MARS proved superior to leading sampling-based replanning algorithms in the state-of-the-art in terms of both success rate and quality of solutions found. The superiority of MARS lies in the fact that it reduces the complexity of the search by trying to connect the current path to a node of the other available paths closer to the current path than the goal.

We also propose a multithread architecture for executing the robot trajectory with continuous replanning and collision checking, applicable to many replanning algorithms. The open-source C++ code of the algorithms and the architecture is available at [41].

Future developments include analysis of how the initial set of paths affects MARS. The trade-off between diversity and quality of initial paths affects the performance, *e.g.*, optimizing the initial path may reduce diversity, with the risk that a single obstacle will block them all, thus reducing the chances of success. Furthermore, we will improve the graph search algorithm. Currently, a depth-first search algorithm is

used to find all solutions with a cost less than the desired value and then extract the valid and less costly one. An alternative and more efficient approach could be taken by LPA* [6], which could avoid searching all solutions before checking the validity of the less expensive one.

REFERENCES

[1] H. S. Hewawasam, M. Y. Ibrahim, and G. K. Appuhamillage, "Past, present and future of path-planning algorithms for mobile robot navigation in dynamic environments," *IEEE Open J. Ind. Electron. Soc.*, vol. 3, pp. 353–365, 2022.

[2] O. Nocentini, L. Fiorini, G. Acerbi, A. Sorrentino, G. Mancioppi, and F. Cavallo, "A survey of behavioral models for social robots," *Robotics*, vol. 8, no. 3, p. 54, Jul. 2019.

[3] G. Michalos, P. Karagiannis, N. Dimitropoulos, D. Andronas, and S. Makris, *Human Robot Collaboration in Industrial Environments*. Cham, Switzerland: Springer, 2022, pp. 17–39.

[4] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Informed sampling for asymptotically optimal path planning," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 966–984, Aug. 2018.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.

[6] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A∗," *Artif. Intell.*, vol. 155, nos. 1–2, pp. 93–146, 2004.

[7] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 354–363, Jun. 2005.

[8] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A∗: An anytime, replanning algorithm," in *Proc. Int. Conf. Automated Planning Scheduling (ICAPS)*, Jan. 2005, pp. 262–271.

[9] S. Aine and M. Likhachev, "Truncated incremental search," *Artif. Intell.*, vol. 234, pp. 49–77, May 2016.

[10] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *Int. J. Robot. Res.*, vol. 21, no. 12, pp. 1031–1052, Dec. 2002.

[11] G. Chiriatti, G. Palmieri, C. Scoccia, M. C. Palpacelli, and M. Callegari, "Adaptive obstacle avoidance for a class of collaborative robots," *Machines*, vol. 9, no. 6, p. 113, Jun. 2021.

[12] H. Liu, D. Qu, F. Xu, Z. Du, K. Jia, J. Song, and M. Liu, "Real-time and efficient collision avoidance planning approach for safe human–robot interaction," *J. Intell. Robot. Syst.*, vol. 105, no. 4, p. 93, Aug. 2022.

[13] Y. Li, X. Hao, Y. She, S. Li, and M. Yu, "Constrained motion planning of free-float dual-arm space manipulator via deep reinforcement learning," *Aerosp. Sci. Technol.*, vol. 109, Feb. 2021, Art. no. 106446.

[14] G. Nicola and S. Ghidoni, "Deep reinforcement learning for motion planning in human robot cooperative scenarios," in *Proc. 26th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2021, pp. 1–7.

[15] J. Bruce and M. M. Veloso, "Real-time randomized path planning for robot navigation," in *Proc. Robot Soccer World Cup*, in Lecture Notes in Artificial Intelligence: Subseries of Lecture Notes in Computer Science, vol. 2752, 2003, pp. 288–295.

[16] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2006, pp. 1243–1248.

[17] D. Ferguson and A. Stentz, "Anytime, dynamic planning in high-dimensional search spaces," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Apr. 2007, pp. 1310–1315.

[18] W. Sun, S. Patil, and R. Alterovitz, "High-frequency replanning under uncertainty using parallel sampling-based motion planning," *IEEE Trans. Robot.*, vol. 31, no. 1, pp. 104–116, Feb. 2015.

[19] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Apr. 2007, pp. 1603–1609.

[20] M. Otte and E. Frazzoli, "RRT^X: Real-time motion planning/replanning for environments with unpredictable obstacles," *Int. J. Robot. Res.*, vol. 35, no. 7, pp. 797–822, 2016.

[21] C. Yuan, G. Liu, W. Zhang, and X. Pan, "An efficient RRT cache method in dynamic environments for path planning," *Robot. Auto. Syst.*, vol. 131, Sep. 2020, Art. no. 103595.

[22] B. Chandler and M. A. Goodrich, "Online RRT* and online FMT*: Rapid replanning with dynamic cost," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 6313–6318.

[23] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2004, pp. 4399–4404.

[24] M. Huppi, L. Bartolomei, R. Mascaro, and M. Chli, "T-PRM: Temporal probabilistic roadmap for path planning in dynamic environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 10320–10327.

[25] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Dept. Comput. Sci., Iowa State Univ., Ames, IA, USA, 1998.

[26] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

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

[28] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT*," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 1478–1483.

[29] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 5369–5375.

[30] 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 *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2014, pp. 2997–3004.

[31] D. Connell and H. M. La, "Dynamic path planning and replanning for mobile robots using RRT," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2017, pp. 1429–1434.

[32] T.-Y. Li and Y.-C. Shie, "An incremental learning approach to motion planning with roadmap management," *J. Inf. Sci. Eng.*, vol. 23, no. 2, pp. 3411–3416, 2002.

[33] Z. Zhang, B. Qiao, W. Zhao, and X. Chen, "A predictive path planning algorithm for mobile robot in dynamic environments based on rapidly exploring random tree," *Arabian J. Sci. Eng.*, vol. 46, no. 9, pp. 8223–8232, Sep. 2021.

[34] Y. Chen, Z. He, and S. Li, "Horizon-based lazy optimal RRT for fast, efficient replanning in dynamic environment," *Auto. Robots*, vol. 43, no. 8, pp. 2271–2292, Dec. 2019.

[35] J. van den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2006, pp. 2366–2371.

[36] J. Vannoy and J. Xiao, "Real-time adaptive motion planning (RAMP) of mobile manipulators in dynamic environments with unforeseen changes," *IEEE Trans. Robot.*, vol. 24, no. 5, pp. 1199–1212, Oct. 2008.

[37] E. Huang, M. Mukadam, Z. Liu, and B. Boots, "Motion planning with graph-based trajectories and Gaussian process inference," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 5591–5598.

[38] K. Kolur, S. Chintalapudi, B. Boots, and M. Mukadam, "Online motion planning over multiple homotopy classes with Gaussian process inference," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 2358–2364.

[39] C. Tonola, M. Faroni, N. Pedrocchi, and M. Beschi, "Anytime informed path re-planning and optimization for robots in changing environments," 2021, *arXiv:2103.13245*.

[40] C. Tonola, M. Faroni, N. Pedrocchi, and M. Beschi, "Anytime informed path re-planning and optimization for human–robot collaboration," in *Proc. 30th IEEE Int. Conf. Robot Hum. Interact. Commun. (RO-MAN)*, Aug. 2021, pp. 997–1002.

[41] C. Tonola and M. Beschi. *An Open-Source Library for Robot Path Replanning*. Accessed: Jan. 9, 2023. [Online]. Available: https://github.com/JRL-CARI-CNR-UNIBS/replanning_strategies.git

[42] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, vol. 2, San Francisco, CA, USA, Apr. 2000, pp. 995–1001.

[43] M.-C. Kim and J.-B. Song, "Informed RRT* with improved converging rate by adopting wrapping procedure," *Intell. Service Robot.*, vol. 11, no. 1, pp. 53–60, Jan. 2018.

[44] M. Faroni, M. Beschi, and N. Pedrocchi, "Safety-aware time-optimal motion planning with uncertain human state estimation," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 12219–12226, Oct. 2022.

[45] M. Faroni and M. Beschi. *SSM_Safety*. Accessed: Jan. 9, 2023. [Online]. Available: https://github.com/CNR-STIIMA-IRAS/ssm_safety.git

CESARE TONOLA received the B.S. and M.S. degrees in industrial automation engineering from the University of Brescia, Italy, in 2018 and 2020, respectively. He is currently pursuing the joint Ph.D. degree in mechanical and industrial engineering with the University of Brescia and National Research Council, Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing, Milan, Italy. His research interests include motion planning and replanning, control techniques for industrial manipulators, and human–robot collaboration.

MARCO FARONI received the B.S. and M.S. degrees in industrial automation engineering and the Ph.D. degree in mechanical and industrial engineering from the University of Brescia, Italy, in 2013, 2015, and 2019, respectively. He was a Researcher with the Italian National Research Council, Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing, Milan, Italy, between 2019 and 2022. He is currently a Research Fellow of the Department of Robotics, University of Michigan.

MANUEL BESCHI (Member, IEEE) received the B.S. and M.S. degrees in industrial automation engineering from the University of Brescia, Italy, in 2008 and 2010, respectively, and the Ph.D. degree in computer science, engineering and control systems technologies from the Department of Mechanical and Industrial Engineering. He has been an Associate Professor with the University of Brescia, since 2022.

NICOLA PEDROCCHI received the M.S. degree in mechanical engineering from the University of Brescia, Italy, in 2004, and the Ph.D. degree in applied mechanics and robotics, in 2008. From 2008 to 2011, he was a Research Assistant with the Institute of Industrial Technologies and Automation, National Research Council of Italy, where he has been a Full Researcher, since 2011. His research interests include control techniques for industrial manipulators in advanced application requiring the interaction robot-environment (e.g., technological tasks) or robot–human operator (e.g., workspace sharing and teach-by-demonstration). He is involved in researches for accurate elastic modeling and dynamic calibration of industrial robots. Since 2015, he has been coordinating the activity of the Robot Motion Control and Robotized Processes Laboratory, CNR-STIIMA.

Open Access funding provided by 'Università degli Studi di Brescia' within the CRUI CARE Agreement