UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DOTTORATO DI RICERCA IN INGEGNERIA MECCANICA E INDUSTRIALE
ING-INF/04 AUTOMATICA
XXXVI CICLO

Reactive Motion Replanning For Human-Robot Collaboration

Ph.D. Candidate:
Cesare Tonola

Ph.D. Supervisor:
Prof. Manuel Beschi

Ph.D. Co-supervisors:
Dr. Nicola Pedrocchi
Dr. Marco Faroni

Academic year 2022/2023

Abstract

In recent years, there has been a significant increase in robots sharing workspace with human operators, combining the speed and precision inherent to robots with human adaptability and intelligence. However, this integration has introduced new challenges in terms of safety and collaborative efficiency. Robots now need to swiftly adjust to dynamic changes in their environment, such as the movements of operators, altering their path in real-time to avoid collisions, ideally without any disruptions. Moreover, in human-robot collaborations, replanned trajectories should adhere to safety protocols, preventing safety-induced slowdowns or stops caused by the robot's proximity to the operator. In this context, quickly providing high-quality solutions is crucial for ensuring the robot's responsiveness. Conventional replanning techniques often fall short in complex environments, especially for robots with numerous degrees of freedom contending with sizable obstacles.

This thesis tackles these challenges by introducing a novel sampling-based path replanning algorithm tailored for robotic manipulators. This approach exploits pre-computed paths to generate new solutions in a few hundred milliseconds. Additionally, it integrates a cost function that steers the algorithm towards solutions that comply with the ISO/TS 15066 safety standard, thereby minimizing the need for safety interventions and fostering efficient cooperation between humans and robots. Furthermore, an architecture for managing the replanning process during the execution of the robot's motion is introduced. Finally, a software tool is presented to streamline the implementation and testing of path replanning algorithms. Simulations and experiments conducted on real robots demonstrate the superior performance of the proposed method compared to other popular techniques.

Sommario

Negli ultimi anni si è assistito a un incremento significativo di robot che condividono lo spazio di lavoro con operatori umani, per combinare la rapidità e la precisione proprie dei robot con l'adattabilità e l'intelligenza umana. Tuttavia, questa integrazione ha introdotto nuove sfide in termini di sicurezza ed efficienza della collaborazione. I robot devono essere in grado di adattarsi prontamente ai cambiamenti nell'ambiente circostante, come i movimenti degli operatori, adeguando in tempo reale il loro percorso per evitare collisioni, preferibilmente senza interruzioni. Inoltre, nelle operazioni di collaborazione tra uomo e robot, le traiettorie ripianificate devono rispettare i protocolli di sicurezza, al fine di evitare rallentamenti e fermate dovute alla prossimità eccessiva del robot all'operatore. In questo contesto è fondamentale fornire soluzioni di alta qualità in tempi rapidi per garantire la reattività del robot. Le tecniche di ripianificazione tradizionali tendono a faticare in ambienti complessi, soprattutto quando si tratta di robot con molti gradi di libertà e ostacoli di dimensioni considerevoli.

La presente tesi affronta queste sfide proponendo un nuovo algoritmo sampling-based di ripianificazione del percorso per manipolatori robotici. Questo approccio sfrutta percorsi pre-calcolati per generare rapidamente nuove soluzioni in poche centinaia di millisecondi. Inoltre, incorpora una funzione di costo che guida l'algoritmo verso soluzioni conformi allo standard di sicurezza ISO/TS 15066, riducendo così gli interventi di sicurezza e promuovendo una cooperazione efficiente tra uomo e robot. Viene inoltre presentata un'architettura per gestire il processo di ripianificazione durante il moto del robot. Infine, viene introdotto uno strumento software che semplifica l'implementazione e il testing degli algoritmi di ripianificazione del percorso. Simulazioni ed esperimenti condotti su robot reali dimostrano le prestazioni superiori del metodo proposto rispetto ad altre tecniche popolari.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

Symbols:

$q, \dot{q}$	robot's joints configuration and speed
q_{current}	current robot configuration
$q_{\text{start}}, q_{\text{goal}}$	start and goal configurations
$\mathcal{C}, \mathcal{C}_{\text{free}}, \mathcal{C}_{\text{obs}}$	configuration space, configuration-free space, and configuration space colliding with obstacles
$\mathcal{CT}$	configuration-time space
$\mathcal{T}, \mathcal{T}_s$	a tree and a subtree
$\mathcal{G}$	a graph
σ	a path
σ^*	optimal path w.r.t. a specified cost function
$w_\sigma = (q_1 \dots q_\mathcal{M})$	sequence of $\mathcal{M}$ waypoints (nodes) of path σ
$\sigma[q_j, q_k]$	portion of σ from $q_j \in w_\sigma$ to $q_k \in w_\sigma$
$\overline{q_i q_{i+1}}$	the i-th connection of σ, where $q_i, q_{i+1} \in w_\sigma$
Ω	set of all paths of a path planning problem
Σ	subset of Ω containing only the feasible paths
$\mathcal{I}$	admissible informed set
t	time instant
τ	a trajectory associated with path σ respecting the kinodynamic constraints of the robot
λ_i	average scaling factor of the i-th connection of σ

Functions & Operators:

$\sigma_i \cup \sigma_j : \Omega \times \Omega \to \Omega$	a function that concatenates σ_j with σ_i (being $\sigma_i(1) = \sigma_j(0)$)

$c(\cdot) : \Omega \to \mathbb{R}_{\geq 0}$	a cost function that associates a positive real cost to a feasible path and $+\infty$ if the path is infeasible
$\|\cdot\|_p : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$	p norm operator, with $p = 2$ if not specified
$\|\sigma\| : \Omega \to \mathbb{R}_{\geq 0}$	length of a path σ
$v \oslash u : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$	component-wise ratio between v and u

Acronyms:

HRC	Human-Robot Collaboration
DoF	Degree Of Freedom of the robot
SSM	Speed and Separation Monitoring
PFL	Power and Force Limiting
APF	Artifical Potential Field
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
ANN	Artificial Neural Networks
MARS	Multi-pAth Replanning Strategy
MARSHA	Human-Aware Multi-pAth Replanning Strategy
RRT	Rapidly exploring Random Trees
RRT*	Rapidly exploring Random Trees Star
PRM	Probabilistic RoadMaps
DRM	Dynamic RoadMaps
DRRT	Dynamic Rapidly exploring Random Trees
ADRRT	Anytime Dynamic Rapidly exploring Random Trees
MPRRT	Multiple Parallel Rapidly exploring Random Trees
HAMP	Human-Aware Motion Planning

CHAPTER 1

Preface

In recent years, robots have gained substantial importance across various industries like automotive, manufacturing, and healthcare. Their capability to automate tasks, enhance production efficiency, and reduce costs has propelled their significance. Initially, robots operated in isolated environments to maximise efficiency without endangering human workers. However, a paradigm shift has occurred, emphasising the importance of collaboration between humans and robots. This trend stems from maintaining adaptability and efficiency even in complex scenarios, merging machine precision and speed with human intelligence and flexibility.

This shift has led to collaborative robots, commonly referred to as cobots, designed to work alongside humans in shared workspaces. Collaborative robots are equipped with advanced sensors and control algorithms to ensure safe interactions with humans, minimising the risk of accidents and enhancing production flexibility. The integration of cobots into industries brings forth a multitude of benefits, such as streamlining production processes, reducing labour expenses, improving product

quality, and increasing overall productivity. However, as the barriers separating robot workspaces are removed, work efficiency and human safety challenges come to the forefront. The assumption that the work environment remains unchanged is no longer valid. In the past, when workspaces were static and pre-defined, robots could move smoothly without interruptions. In contrast, today's dynamic workspaces, which include moving obstacles like human operators, often impede the robot's movement. Robots must either slow down or stop whenever an operator gets close to prevent collisions. This behaviour can be inefficient, mainly if the operator frequents the area or remains there for prolonged periods.

This problem can be addressed at the motion planning level. The idea is to modify the calculated path on-the-fly to adapt to environmental changes and avoid collisions. To tackle this challenge, researchers are actively exploring motion replanning techniques. Generating an entirely new path whenever a new obstacle appears requires substantial computational resources and time. A more efficient approach involves dynamic adjustment of the existing path to quickly and efficiently respond to changes. Nevertheless, existing techniques still require significant computational power, impacting their responsiveness.

Replanning algorithms need to be supported by complex software architectures. When dealing with unchanging environments, the regular approach to motion planning involves computing a path, assigning it a time parametrisation, and then executing the resulting trajectory. On the other hand, when it comes to motion replanning, there is a greater demand throughout the trajectory's execution. It is essential to consistently reevaluate and adjust the plan as the robot moves, smoothly transitioning to new trajectories. This calls for an architecture that not only supervises the robot's motion execution but also maintains up-to-date scene information while managing the replanning process simultaneously.

However, in workspaces shared with humans, it is necessary to do

more than ensure that the robot's path is collision-free. The robot's speed adjustments and halts are controlled by safety systems that continually monitor the workspace to guarantee human safety. The ISO/TS 15066 safety standard proposes *collaborative operations* like *Speed and Separation Monitoring* (SSM) or *Power and Force Limiting* (PFL) to adhere to safety regulations and protocols. The choice of strategy depends on risk analysis and aims to reduce potential hazards. In this context, the robot should work minimising the need for safety interventions to reduce wasted time and enhance collaborative productivity. To achieve this goal, implementing adaptive strategies across task planning, motion planning, and control levels is imperative. Building upon the existing safety module, these strategies proactively and reactively oversee the robot's actions, ensuring smooth cooperation with the operator in shared spaces.

Therefore, the conventional approach to path replanning needs to be reconsidered. Simply ensuring a collision-free trajectory does not necessarily translate to a safety-compliant robot's behaviour. Modifying the path to avoid collisions might cause the safety system to intervene significantly, resulting in slowdowns or stops, which could undermine the robot's effectiveness. As a result, replanning algorithms need to be aware of the effects of the safety module. These algorithms should consider humans not just as obstacles but as factors that affect how the trajectory should be modified to achieve efficient behaviour. Unfortunately, this adds computational complexity, which can impact robot promptness.

A promising way to address the challenges of motion replanning is through the sampling-based approach, which has become increasingly popular in recent decades for solving path planning problems. This method seeks for solutions by randomly sampling the robot search space and it is favored for its simplicity, adaptability, and ability to scale, even when dealing with large search spaces (*e.g.*, robots with many degrees of freedom). However, as the number of dimensions increases, finding

new solutions becomes more complex. This results in a greater number of algorithm iterations, potentially burdening computations and slowing down the algorithm's responsiveness. Consequently, reactive path replanning for robots with numerous degrees of freedom (*e.g.*, robot manipulators) remains a challenge that is yet to be fully solved.

This thesis aims to address the aforementioned problems by introducing a novel algorithm of path replanning for many degrees of freedom robots. In this regard, the following contributions are presented:

- A framework to manage the replanning process. A software architecture is devised to oversee the execution of the robot's trajectory, continuous scene updates, and concurrent replanning. This framework is adaptable to a wide range of sampling-based path replanning algorithms. Importantly, it alleviates the algorithm's burden of collision checking and cost updating of the current path.

- A sampling-based multi-path replanning algorithm. The algorithm is designed for fast and efficient path replanning, even when dealing with robots with numerous degrees of freedom. Instead of aiming to find an entirely new solution to reach the goal, the algorithm focuses on connecting to pre-existing paths that lead to the same goal. This approach accelerates the initial solution discovery process, which can subsequently be refined over time. Formal guarantees about the completeness and optimality of the method are provided, ensuring its reliability.

- A safety-aware cost function for collaborative applications. A cost function is designed to guide the replanning algorithm towards generating solutions that prioritize safety compliance. By minimizing the necessity for safety module interventions, it promotes an efficient robot behavior. The resulting algorithm facilitates improved human-robot cooperation, enabling the robot to efficiently adjust

its motion in response to unexpected movements of the operator in unstructured tasks and environments.

- A flexible software tool to solve path replanning problems. The core of this tool is a library that streamlines motion replanning challenges with minimal code requirements. It serves as a valuable resource for researchers aiming to develop new algorithms while simplifying the practical application of solutions.

Proposed algorithms are evaluated through comprehensive comparisons with state-of-the-art approaches. A combination of simulations and real-world experiments was conducted to assess the suitability of the proposed approach in shared workspace scenarios. These tests highlight its performance in unstructured scenarios and aim to show in which contexts it provides advantages over standard solutions.

The dissertation is structured as follows: Chapter 2 presents path planning and replanning problems along with the key techniques employed to address them. Chapter 3 introduces an architecture that enables simultaneous trajectory execution and online path modification, allowing the robot to adapt its trajectory without interruption. The proposed sampling-based path replanning algorithm is detailed in Chapter 4, while Chapter 5 covers its integration with a safety-aware cost function. Chapter 6 introduces an open-source tool designed to streamline the process of implementing, testing, and utilizing replanning algorithms. Finally, Chapter 7 provides the concluding remarks.

CHAPTER 2

Path planning and replanning problems

This chapter provides a comprehensive overview of path planning and replanning techniques. It begins by elucidating the differences between path planning and replanning problems. The subsequent section delves into the prominent strategies found in existing literature for addressing path planning challenges and how these strategies are adapted to handle replanning scenarios. It gives special attention to the advantages of employing a sampling-based methodology and the attendant challenges in this context. Additionally, the chapter highlights the requisites for efficient motion planning in human-robot collaboration. Finally, Section 5.1.1 delineates the specific contributions of this thesis.

2.1 Introduction

One of the primary objectives in the field of robotics is to make robots fully autonomous. This means that robots should be able to understand high-level commands and complete desired tasks without any human

intervention. The concept of enabling robots to autonomously understand and execute provided instructions has captured significant attention across diverse sectors, spanning industries, space and maritime exploration, search and rescue missions, medical applications, and social services [92]. A pivotal component in achieving this autonomy is *motion planning*. The motion planning problem is one of the most studied in robotics [36]. It consists of determining the sequence of movements a robot must perform to successfully complete a designated task, such as moving from a starting point to a desired endpoint.

Generally, this problem involves two main steps. The first one, known as *path planning*, focuses on determining the sequence of points the robot needs to reach to accomplish the desired task. The second step is the *time parameterisation*, aimed at computing the *timing-law* for traversing the path. This computation results in a trajectory encompassing both the geometric path to be followed and the corresponding velocity profile. The trajectory essentially consists of positions, velocities, and accelerations over time, dictating where and when to move the robot. This sequence of information is then sent to the robot's low-level controller to execute the entire motion.

Path planning and *trajectory planning* address two separate problems. Path planning focuses on determining the route from a starting point to a goal point, in the robot's configuration space or in the Cartesian space. This path must be *feasible*, meaning the robot can execute it without collisions or violating system constraints. Furthermore, the path should fulfil specific task requirements, such as being the shortest one.

On the other hand, trajectory planning deals with finding the best way to move along a selected path, considering the robot's kinematic and dynamic constraints while simultaneously adhering to various criteria, such as minimizing energy [14], execution time [129], and jerk [130].

Although a motion is fully defined only once the trajectory is deter-

mined, this dissertation primarily focuses on paths, relying on state-of-the-art techniques for time parametrisation.

In addition, enabling autonomous movement for robots necessitates ensuring their capability to move even in open and unpredictable environments, where they are not the sole moving entity. This imperative applies to all the previously mentioned sectors, encompassing industrial realm.

Initially, in industrial settings, robots operated within confined cells with known and static surroundings, devoid of unfamiliar moving objects. In this context, the robot's motion is calculated to satisfy various criteria, such as optimizing productivity, speed, and energy efficiency, while adhering to system limitations, workspace constraints, and task prerequisites. However, computing these motion trajectories incurs substantial computational costs and time consumption. Nonetheless, if tasks are repetitive and consistently demand the same motions, these trajectories can be computed beforehand. This methodology, known as *offline planning*, rests on the presumption of a static environment or foreknowledge of potential changes. It does not account for unexpected environmental shifts demanding the robot to adapt on the fly. Consequently, this approach find applicability in well-defined and controlled environments.

In contrast, offline planning proves inadequate in dynamic environments, where a robot must adapt promptly to its surroundings. Achieving this capacity entails proficiency in environmental sensing, decision-making, and adaptive motion. However, this thesis specifically focuses on the latter aspect – modifying the robot's trajectory during the motion. This can involve actions like slowing down [39] or altering the path [10]. Commonly referred to as *online planning*, this approach is carried out during motion and cannot be precalculated since it responds to unexpected events. Notably, to react swiftly to environmental changes, the robot's response must be computed rapidly. While velocity scaling strategies can

usually be applied in each cycle of task execution, path replanning techniques require more computational resources, often taking tens to hundreds of milliseconds to find a new solution. These strategies alter the robot's path during execution to evade collisions and meet specific task criteria. These types of algorithms constitute the focus of this thesis, with particular regard to path replanning strategies applied to **robotic manipulators**.

2.2 Problem formulation

Although the robot physically operates within its workspace, the path planning problem is typically described using the concept of the robot's *configuration space* [105], denoted as $\mathcal{C}$. Definition 1 provides the formulation of the problem.

Definition 1: The Feasible Path Planning Problem

Consider the configuration space of the robot, denoted as $\mathcal{C} \subseteq \mathbb{R}^n$, as the search space for the path planning problem. Here, n signifies the number of degrees of freedom (DoF) of the robot, while $q \in \mathcal{C}$ represents the generalized coordinates of the system. In the context of robot manipulators, q typically corresponds to a real-valued vector of joint positions. The subset $\mathcal{C}_{\text{obs}} \subset \mathcal{C}$ comprises configurations that lead to collisions with obstacles. Defined $\text{cl}(\cdot)$ as the closure of a set, $\mathcal{C}_{\text{free}} = \text{cl}(\mathcal{C} \setminus \mathcal{C}_{\text{obs}})$ is the set of collision-free configurations.

Given an initial configuration $q_{\text{start}} \in \mathcal{C}_{\text{free}}$ and a goal configuration $q_{\text{goal}} \in \mathcal{C}_{\text{free}}$, solving the path planning problem involves determining a curve $\sigma : [0, 1] \to \mathcal{C}$, namely a path, such that $\sigma(0) = q_{\text{start}}$ and $\sigma(1) = q_{\text{goal}}$. A curve that lies entirely in $\mathcal{C}_{\text{free}}$ is referred to as a *feasible path*, while one that intersects $\mathcal{C}_{\text{obs}}$ is termed an *infeasible path*.

Note that the goal may not be a single configuration q_{goal} but a set of desired configurations $\mathcal{Q}_{\text{goal}} \subset \mathcal{C}_{\text{free}}$.

Furthermore, many applications necessitate a feasible path that fulfills particular requirements. For instance, you could be searching for the shortest path. Hence, path planners typically seek for a feasible path that optimizes a desired objective. Definition 2 presents the optimal path planning problem.

Definition 2: The Optimal Path Planning Problem

Let Ω be the set comprising all paths, both feasible and unfeasible, that connect $q_{\text{start}} \in \mathcal{C}_{\text{free}}$ to $q_{\text{goal}} \in \mathcal{C}_{\text{free}}$. Define $\Sigma \subset \Omega$ as the subset containing only feasible solutions. Paths are assessed using a cost function $c : \Omega \to \mathbb{R}_{\geq 0}$, which assigns a non-negative real number as the cost to any feasible path $\sigma \in \Sigma$, and assigns infinity to any infeasible path.

An *optimal path* is a feasible path σ^* that satisfies the following condition:

$$\sigma^* = \underset{\sigma \in \Sigma}{\operatorname{argmin}} \{c(\sigma) | \sigma(0) = q_{\text{start}}, \sigma(1) = q_{\text{goal}}\} \tag{2.1}$$

One of the most simple and popular cost function $c(\sigma)$ measures the length of the path, so that the optimal solution is the shortest collision-free path from q_{start} to q_{goal}.

Depending on whether the planner is able to solve the problems defined by Definitions 1 and 2 or not, it can be classified as *complete* and/or *optimal*. A path planner is said to be *complete* if it ensures finding a solution in a finite amount of time, if one exists. Furthermore, if it also guarantees finding the optimal solution, the planner is referred to as *optimal* [94]. By definition, the optimality of the planner implies its completeness. Completeness and optimality are fundamental characteristics of

path planning algorithms. However, currently, there is no algorithm that possesses these properties. Instead, algorithms typically offer weaker notions, like *resolution* or *probabilistic completeness* and *resolution* or *almost-sure asymptotic optimality*. Further details regarding these properties will be provided in Section 2.3.

It is also important to distinguish *multi-query* planning scenarios from *single-query* planning scenarios. In the first case, the robot needs to navigate through the same area multiple times, and $\mathcal{C}_{\text{obs}}$ remains relatively constant over time. In such cases, it is beneficial to gather information about the entire search space before any motion occurs. This information can then be utilized to efficiently solve multiple planning problems. This approach requires an initial preprocessing step to comprehensively explore the search space.

On the other hand, in single query planning scenarios, the robot either does not repeatedly navigate the same area or $\mathcal{C}_{\text{obs}}$ changes significantly over time. In this context, exploring the search space beforehand becomes inefficient because the gathered information would quickly become outdated. Consequently, the path planning problem is addressed just before each individual movement [46]. It is important to note that this does not imply the absence of collisions with $\mathcal{C}_{\text{obs}}$, if it evolves unexpectedly during the robot's motion. Handling this aspect of the problem involves online motion planning/replanning.

While there is a standard formulation for motion planning problems shared among researchers, the same cannot be said for replanning problems. Definitions 1 and 2 consider only static obstacles, *i.e.*, $\mathcal{C}_{\text{obs}}$ is constant as the robot moves. Defining a general replanning problem is challenging due to the dynamic nature of real-world environments. Obstacles might move continuously or discretely, and their behavior could range from predictable to unpredictable [167]. Moreover, the robot's working environment can be fully observable, for example by using cameras in a

collaborative industrial cell, or be discovered during the robot's motion, as in the case of mobile robotics. Consequently, formulating a replanning problem in a general way is intricate. In this dissertation, we will focus on the replanning problem closest to the classic path planning problem as in Definition 1. Therefore, the path replanning problem can be formulated as follows:

Definition 3: The Path Replanning Problem

Let $\mathcal{C}_{\text{free}}(t)$ be the subset of all configurations not colliding with obstacles at a specific time instant t. Consider an initial path $\sigma_{\text{init}} : [0,1] \rightarrow \mathcal{C}_{\text{free}}(0)$, with $\sigma_{\text{init}}(0) = q_{\text{start}}$ and $\sigma_{\text{init}}(1) = q_{\text{goal}}$ and let $q_r(\hat{t})$ be the robot configuration at time $\hat{t}$, such that $\sigma_{\text{init}}[q_r(\hat{t}), q_{\text{goal}}]$ is the portion of the path between $q_r(\hat{t})$ and the goal. *Path replanning* at time instant $\hat{t}$ means finding a new path $\sigma_{\text{new}} : [0,1] \rightarrow \mathcal{C}_{\text{free}}(\hat{t})$, with $\sigma_{\text{new}}(0) = q_r(\hat{t})$ and $\sigma_{\text{new}}(1) = q_{\text{goal}}$, such that:

$$c(\sigma_{\text{new}}) < c(\sigma_{\text{init}}[q_r(\hat{t}), q_{\text{goal}}]) \tag{2.2}$$

where $c(\sigma)$ is equal to infinity when σ is not feasible (*e.g.*, in collision with obstacles).

Notably, path replanning can occur either when the existing path becomes obstructed by new obstacles, or when the solver identifies a potentially better path (*i.e.*, with a lower cost). While some algorithms are designed to address only the first situation, more advanced ones can effectively handle both cases.

Furthermore, it is worth noting that the inherent unpredictability of the replanning context complicates the provision of formal guarantees for these algorithms, such as *completeness* and *optimality*. Such properties cannot be conclusively proven for replanners without making assumptions about obstacle dynamics. Counterexamples can be constructed

where an obstacle obstructs any newly proposed solution, preventing the robot from finding a path to its goal. Nevertheless, these properties can be proved for a replanning algorithm if we assume that, starting from time $\hat{t}$, $\mathcal{C}_{\text{free}}$ remains constant.

2.2.1 The configuration-time space

In dynamic environments planning, extending the configuration space to incorporate time as an additional dimension gives rise to what is known as the *configuration-time space*, introduced the first time by Fraichard in [45]. This approach, widely utilized in various strategies for path planning and replanning in non-static environments, allows for the consideration of obstacles movement. As a result, it becomes possible to identify paths that steer clear of intersecting with obstacles precisely when they pass through specific points [167].

Let's define the time interval of interest as $\mathcal{T} = [0, t_{\max}]$, where 0 denotes the initial time. The resulting space $\mathcal{CT} := \mathcal{C} \times \mathcal{T}$ comprises pairs $\langle q, t \rangle$, where $q \in \mathcal{C}$ and t is a scalar between 0 and $t_{\max}$ representing time. If the robot configuration q collides with obstacles at time t, then $\langle q, t \rangle$ belongs to $\mathcal{CT}_{\text{obs}}$. Consequently, both stationary and moving obstacles in the configuration space become static obstacles in the configuration-time space.

In $\mathcal{CT}$, planning entails charting a trajectory. Conventional methods involve searching for a path and calculating the timings needed to reach waypoints, taking into account the robot's maximum velocity constraints. It is important to note that in this space, the goal typically comprises not just a single configuration $\langle q_{\text{goal}}, t \rangle$, but rather a locus defined by q_{goal} and multiple possible t.

The inclusion of time in the search space could enable more informed planning. When dealing with obstacles whose trajectories are known in advance, it is possible to plan a trajectory that avoids intersections. Dur-

ing online planning, if you can predict the obstacle's motion, you can devise a new trajectory that obviates the need for further replanning. For example, there may be no necessity to alter the route if one can accurately predict the imminent removal of the obstacle in the near future. However, it is important to highlight that this approach introduces an additional layer of complexity to the search space. This is attributed to the inclusion of an extra dimension and the necessity for a reliable predictive estimate of the obstacle's motion. This presents a significant challenge, as such estimations are not always readily available. Furthermore, when they are, they tend to be short-term and subject to noise. The high level of uncertainty in these estimations often leads to conservative long-term planning, which can result in either the inability to identify a viable path or the generation of very conservative and therefore inefficient solutions. This is particularly problematic in scenarios involving shared workspaces between humans and robots, where safety and efficiency are paramount objectives. Moreover, within the realm of human-robot cooperation, obtaining dependable estimations proves to be a daunting task, which falls beyond the scope of this thesis. As a result, this thesis places its emphasis on techniques confined to the **configuration space**. Specifically, we aim to develop a reactive algorithm with the capacity to swiftly adapt the path in response to human movements.

However, the upcoming section will explore solutions to the motion replanning problem from the perspectives of both search space formulations.

2.3 Path planning and replanning techniques

This section delves into the primary techniques employed for tackling both path planning and replanning problems. Four distinct categories of approaches commonly used to address motion planning challenges will

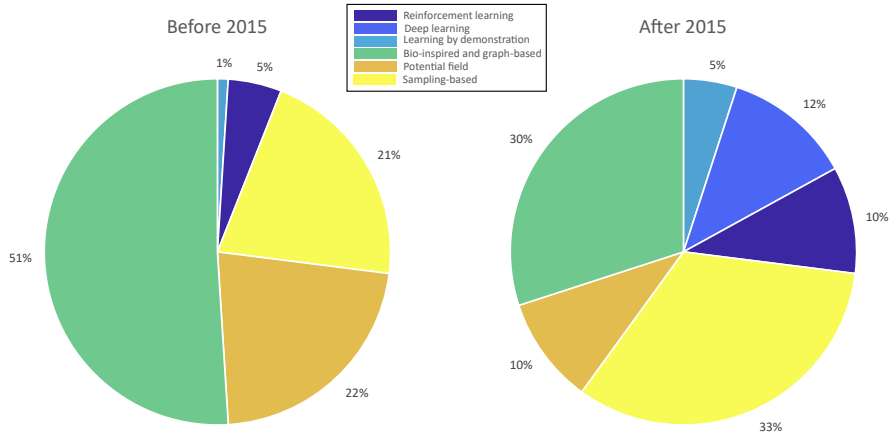


Figure 2.1: Distribution of different path planning approaches before and after 2015 (data retrieved from [158]).

be highlighted, the *sampling-based* approach, the *graph-based* approach, the *optimization-based* approach and the *learning-based* approach [158]. Within these categories, certain methods naturally possess the ability to address path replanning problems from the outset. On the other hand, for other approaches, the algorithms designed for path replanning are developed as an extension of the existing path planning solutions.

Figure 2.1 shows the distribution of the different approaches before and after 2015 [158]. Notably, the sampling-based approach remains the most prevalent, even though the popularity of certain learning-based methods has grown in recent years. This enduring prevalence can be attributed to its relative simplicity, efficient scalability even with expanding search spaces, inherent flexibility, and the added advantage of not necessitating a direct representation of the search space. It is important to emphasize that the emerging artificial intelligence strategies often complement sampling-based algorithms [67, 140, 178, 193]. Consequently, it is reasonable to expect that sampling-based methodology will remain

highly relevant and popular in the years ahead.

Among the strategies we will discuss, the sampling-based approach will receive the most comprehensive exploration, as it forms the foundational methodology behind the algorithm proposed in this thesis. Conversely, the other strategies will be briefly overviewed.

2.3.1 The optimization-based approach

Optimization-based approaches exploit optimization techniques to find solutions that minimize a specific objective function, typically related to collision avoidance and trajectory smoothness. The problem is usually formulated as follows:

$$\min_{q_1,q_2,\dots,q_N} \sum_{i=1}^{N} c(q_i) + \frac{1}{2} \sum_{j=1}^{n} ||Aq^j||^2 \tag{2.3}$$

Here, $q_i \in \mathbb{R}^n$ is the i-th point of the trajectory, $q^j \in \mathbb{R}^N$ represents the trajectory of the j-th joint, N the number of points in the discretized trajectory and n the number of DoF. The function $c(\cdot)$ is an arbitrary state-dependent cost function that may encompasses terms related to obstacles avoidance, joints limits and torques constraints. The matrix $A \in \mathbb{R}^{N \times N}$ is employed to approximate dynamic quantities through finite differencing:

$$A = \begin{pmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

Consequently, $\sum_{j=1}^{n} ||Aq^j||^2$ approximates the second order derivative of the trajectory and is minimized to reduce accelerations and attain smooth solutions.

Optimization-based approaches mainly differ on how they solve the problem of Equation (2.3). They are generally categorized into *gradient-based optimizations* and *stochastic optimizations*. One well-known example of a gradient-based optimization algorithm is CHOMP (Covariant Hamiltonian Optimization and Motion Planning) [142]. It starts with an initial, possible infeasible, trajectory and employs a covariant gradient descent technique to find a smooth, collision-free, trajectory through the configuration space between two defined end points. CHOMP has been applied in various scenarios, including 6-DoF robotic arms and walking quadruped robots, and has been integrated into numerous motion planning algorithms [100, 141, 172]. However, it relies on the cost function being smooth and differentiable, which is not always the case, especially when dealing with motor torques constraints.

To address this limitation, STOMP (Stochastic Trajectory Optimization for Motion Planning) [72] uses an approach that does not require knowledge of the cost function's gradient. It generates noisy trajectories to explore the space around an initial, possibly infeasible trajectory and combines these to approximate the gradient and achieve a lower-cost solution.

Nevertheless, gradient-based methods may become trapped in local minima, impeding the discovery of the optimal solution.

Further optimization-based approaches have been developed trying to solve the local minima problem and speed up the convergence towards the global optimum. Among the main strategies, Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) are popular solutions [1, 62, 79, 136, 176]. Generally, stochastic optimization methods are better at avoiding local minima and improving planning efficiency than gradient-

based methods due to their capacity to explore a wider solution space.

Moreover, ongoing advancements in numerical optimization algorithms have significantly improved the computational efficiency of optimization-based algorithms, making them more widely adopted due to their enhanced ability to search for optimal solutions.

The aforementioned approaches are mainly designed for applications with static environment. However, such strategies have been used as the basis for online planning algorithms capable of adapting the original plan to changes in the environment.

ITOMP (Incremental Trajectory Optimization for Real-Time Replanning in Dynamic Environments) [124] aims to address motion planning in dynamic environments using an optimization-based approach. This method does not distinguish between an initial planning phase and a replanning phase during the execution of the trajectory. Instead, it adopts an incremental approach to plan in real-time. At each iteration it estimates the trajectory of the obstacles in a limited time horizon. Then, it computes a trajectory from the robot configuration to a final configuration by solving the optimization problem of Equation (2.3) to avoid collisions and satisfy trajectory smoothness and torque requirements. This algorithm operates within a framework that interleaves planning with execution, allocating a time budget for planning. The solver is interrupted when the time limit is reached, a strategy necessitated by the need for rapid responses and the limited time validity of obstacle trajectory information. This architectural approach has been leveraged in several other works and extended to accommodate varying planning and execution time intervals [59, 119, 125, 127]. While the solution found may be suboptimal and not satisfy all requirements, it is recursively employed as input for the optimization in the next iteration, leading to potentially improved results more quickly. Additionally, [125] introduces GPU-based parallel processing to accelerate ITOMP.

PTOMP (Progressive Trajectory Optimization-based Motion Planning algorithm for Human-Robot Interaction Scenarios) [119] builds upon the ITOMP framework by updating online the parameter of the optimization process and integrating an interaction-oriented cost function. These advancements are designed to facilitate more seamless and natural interactions within of Human-Robot Interaction (HRI) scenarios.

RAMP (Real-Time Adaptive Motion Planning) [169] tackles online motion planning by drawing from evolutionary computation. It still employs a planning and execution framework but plans trajectories in the $\mathcal{CT}$ space. The algorithm maintains a population of trajectories and improves their quality (fitness) at each planning time using random operators (such as inserting, deleting, or swapping trajectory points). RAMP is demonstrated to effectively handle drastic changes in the environment through global planning of diverse trajectories. Another real-time motion planner based on genetic optimization is presented in [168].

In broad terms, optimization-based approaches offer the flexibility to seamlessly incorporate new constraints into the planning process by adjusting the cost function. Nevertheless, it is important to note that these methods come with certain drawbacks: they tend to be computationally demanding, lack assurances of reaching the global optimum, and frequently necessitate to predefine the trajectory duration.

2.3.1.1 The Artificial Potential Field approach

The Artificial Potential Field (APF) method is a widely employed technique in the field of motion planning. Its popularity stems from its intuitive concept, simplicity, lightweight nature, and excellent real-time performance. Consequently, it finds extensive application in real-time obstacle avoidance algorithms. It deviates from the optimization-based algorithms discussed earlier, adopting a distinct strategy while still incorporating a *gradient descent* approach.

Originally proposed by O. Khatib in 1985 [78], the core idea behind APF is to guide a robot through an artificially generated potential field. This field creates attractive forces towards the goal and repulsive forces around obstacles, ultimately determining the direction and magnitude of the robot's velocity vector.

The foundation of this approach is the concept of the *operational space*, which defines a coordinate system where x represents a set of independent parameters describing the end-effector's position and orientation in a reference frame R_0. Control of the manipulator in the operational space is based on the selection of a command force F applied to the end-effector. In order to produce F, joint torques Γ need to be computed through the relation:

$$\Gamma = J^{\mathsf{T}}(q)F \tag{2.5}$$

Here, $J(q)$ represents the Jacobian matrix for the robot configuration q.

The artificial potential field is computed as the sum of two components:

$$U_{\text{art}}(x) = U_{x_{\text{goal}}}(x) + U_{\text{obs}}(x) \tag{2.6}$$

Which leads to:

$$U(x) = U_{\text{art}}(x) + U_g(x) \tag{2.7}$$

Here, $U_{x_{\text{goal}}}$ is the artificial potential field which attracts the robot to the goal, U_{obs} repels from obstacles and U_g is the potential energy of gravity.

Using the Lagrangian formalism and the end-effector dynamic decoupling equations, the end-effector equation of motion can be written as:

$$\Lambda(x)F^*_{\text{art}} + \mu(x,\dot{x}) + p(x) = F \tag{2.8}$$

$\Lambda(x)$ denotes the kinetic energy matrix, $\mu(x, \dot{x})$ represents the centrifugal and Coriolis forces, and $p(x)$ the gravity forces. $F^*_{\text{art}}(x)$ is the command force vector of the decoupled end-effector which becomes equivalent to a *single-unit mass*. $F^*_{\text{art}}(x)$ can be written as:

$$F^*_{\text{art}}(x) = F^*_{x_{\text{goal}}}(x) + F^*_{\text{obs}}(x) \tag{2.9}$$

where:

$$\begin{aligned} F^*_{x_{\text{goal}}}(x) &= -\text{grad}[U_{x_{\text{goal}}}(x)] \\ F^*_{\text{obs}}(x) &= -\text{grad}[U_{\text{obs}}(x)] \end{aligned} \tag{2.10}$$

$U_{x_{\text{goal}}}(x)$ is typically defined as:

$$U_{x_{\text{goal}}}(x) = \frac{1}{2}k(x - x_{\text{goal}})^2 \tag{2.11}$$

where k represents the position gain. The attractive force $F^*_{x_{\text{goal}}}(x)$ can be derived as:

$$F^*_{x_{\text{goal}}}(x) = -k(x - x_{\text{goal}}) - \xi\dot{x} \tag{2.12}$$

The term $\xi\dot{x}$ is introduced to achieve asymptotic stability of the system.

The $U_{\text{obs}}(x)$ component is designed to satisfy the manipulator stability condition and create a potential barrier at each point on the obstacle's surface. It must be a non-negative, differentiable function and that approaches infinity as the end-effector nears the obstacle's surface. The resulting $U_{\text{art}}(x)$ should have global minimum at $x = x_{\text{goal}}$. To prevent unwanted perturbations beyond the vicinity of the obstacle, the influence of $U_{\text{obs}}(x)$ should be confined to a limited region surrounding the obstacle. A potential description of $U_{\text{obs}}(x)$ is the following:

$$U_{\text{obs}}(x) = \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho} - \frac{1}{\rho_0})^2 & \rho \leq \rho_0 \\ 0 & \rho > \rho_0 \end{cases} \tag{2.13}$$

Where ρ and ρ_0 represent the minimum distance from the obstacle and the limit distance of the potential field's influence, respectively. The resulting force $F_{\text{obs}}(x)$ is defined as:

$$F^*_{\text{obs}}(x) = \begin{cases} \eta(\frac{1}{\rho} - \frac{1}{\rho_0})\frac{1}{\rho^2}\frac{\partial \rho}{\partial x} & \rho \leq \rho_0 \\ 0 & \rho > \rho_0 \end{cases} \tag{2.14}$$

This approach can also be extended to have multiple points on the robot subjected to the potential. [78] provides further details.

The inherent adaptability of this strategy makes it highly suitable for dynamic environments. Extensive research has been devoted to addressing the challenge of local minima [126, 173, 189, 192], and enhancing robot behavior by factoring in the relative speed between obstacles and the robot when defining the potential field [52, 137, 156].

While the APF method enjoys widespread popularity in mobile robotics, it has also found application in more intricate systems like robot manipulators [22, 23, 144, 171, 177]. In this approach, besides directing the motion of the end-effector, repulsive forces are strategically applied to the robot's point nearest to obstacles, effectively pushing it away from potential collision points.

Drawing inspiration from the APF paradigm, the Elastic Band [139] and Elastic Strips [10] frameworks dynamically deform a precomputed trajectory, respectively, in configuration space and operational space, in response to moving obstacles. These frameworks apply two types of forces: a repulsive force from obstacles to avert collisions, and an internal elastic force within the trajectory. Consequently, the trajectory continuously adapts to steer clear of obstacles while maintaining smoothness

and shortening when possible.

However, it is worth noting that while this approach excels at making local adjustments, it may face challenges when acting as a global planner. For instance, if a previously accessible path suddenly becomes obstructed, these algorithms may struggle to find an alternative route to the goal. Additionally, the calculation of repulsive forces is based on a limited number of key-points along the robot's kinematic structure and the obstacles. Consequently, collision avoidance can be assured only for these designated points.

In practice, optimization-based techniques necessitate simplifying intricate obstacles into a series of simpler components, such as smaller spheres. However, for complex robotic environments, this strategy becomes impractical. Attempting to incorporate additional test points on both robots and environmental obstacles can result in heightened computational complexity and greater resource demands.

2.3.2 The learning-based approach

The learning-based category encompasses a range of strategies employing artificial intelligence techniques to address motion planning in both static and dynamic environments. Notably, popular techniques include Artificial Neural Networks (ANN) and Reinforcement Learning (RL).

Motion planning utilizing neural networks is gaining traction due to their ability to model nonlinear functions and their reduced computational complexity during inference [112, 172, 179, 187]. A prominent strategy involves employing neural networks to define trajectory waypoints, whether in joint or Cartesian space, and then establishing smooth connections between them using polynomial functions.

For example, Meziane et al. [112] introduced SPADER, a two-layer neural network capable of generating trajectory waypoints in Cartesian space to move around dynamic obstacles. The robot's motion unfolds in

a set of subspaces within a shared workspace, which are interconnected to obtain smooth movements and prevent collisions with dynamic obstacles, such as a human's limb. Specifically, the neural network generates waypoints in intermediate subspaces necessary for guiding the robot end-effector along a collision-free path. These points are then connected using quintic polynomials to ensure continuity in velocities, accelerations, and minimize abrupt changes. However, it is important to note that since the algorithm operates in Cartesian space, it applies solely to end-effector motion planning and does not guarantee that the robot links will not collide with humans.

Wang et al. [172] employed a Collaborative Waypoint Planning network (CWP-net) to generate all crucial waypoints for dynamic obstacle avoidance in joint space based on environmental data. Quintic polynomials are then used to create a seamless trajectory through these waypoints, adhering to velocity and acceleration constraints. A modified version of STOMP is then utilized to perform localized trajectory optimization in cases where the segments between waypoints, as computed by the CWP-net, lead to a collision.

Duguleana et al. [34] used Q-learning and a double neural network to iteratively plan the motion online and avoid obstacles. As the plan is constructed incrementally, it is not feasible to pre-calculate the trajectory's cost and its smoothness.

Reinforcement Learning stands as another widely adopted technique for manipulator motion planning in dynamic environments. It empowers robots to iteratively learn optimal behaviors through interactions with their surroundings. Unlike conventional approaches that require explicit solutions, RL relies on a scalar objective function to assess the robot's performance in each step [83]. Operating within the framework of Markov decision processes, an agent engages with the environment by perceiving a state, executing an action, transitioning to a subsequent state, and re-

ceiving a reward. This reward signal encapsulates the fundamental objective and furnishes the agent with performance feedback. Consequently, the agent's aim is to acquire a policy that maximizes the overall expected reward function [140].

While previously effective primarily in simple planning problems, RL has recently found applicability in high-dimensional challenges, facilitated by the use of neural networks for estimating the reward function. This approach, known as Deep Reinforcement Learning (DRL), has demonstrated its ability to tackle intricate tasks and solve complex problems. The conventional method involves implementing model-free RL techniques and conducting training phases focused on learning the optimal policy from scratch, without any human guidance, to move the robot while avoiding obstacles [148, 175].

For example, Nicola et al. [114] presented an online motion planning framework leveraging DRL for applications in human-robot cooperation. Notably, they advocate for a formulation where the robot's action is characterized as a trajectory with a duration corresponding to a timestamp. This stands in contrast to conventional methods that typically produce a joint speed command or variations in joint speed. This adjustment is geared towards attaining smoother trajectories. Furthermore, this approach eliminates the necessity to deduce human intentions, discern the task being performed by the human, or predict the specific workspace occupied by the human.

Kamali et al. [74] introduced a dynamic goal DRL approach for effective path planning of a robot arm. Their work proposed an intuitive method for translating human hand motion into corresponding movements of the robot arm with collision avoidance.

Sangiovanni et al. [147] presented a novel approach to obstacle avoidance in robot manipulators, utilizing a hybrid control scheme incorporating DRL. This scheme employs a dual architecture, where an initial plan

is generated using a standard motion planner to accelerate the trajectory creation process. The robot initially adheres to this plan, and upon meeting specific obstacle-distance criteria, the DRL-based control, specialized in collision avoidance, is engaged. This hybrid switching methodology harnesses the strengths of both classical algorithms and DRL-based controls, thereby ensuring a heightened level of stability, particularly when adhering to a predetermined path. Simultaneously, it reduces the need for intricate hand-engineering, especially in scenarios involving complex operations like full-body collision avoidance.

Several studies leverage RL, ANN, and learning-based techniques to support various categories of planners, particularly sampling-based ones. This approach aims to expedite certain planning phases (*e.g.*, the sampling phase, detailed in Section 2.3.4.1) to reduce computation times and achieve faster responses [21, 67, 68, 181, 186, 193]. Learning from experience allows to generate more effective and efficient sample nodes in a way that decreases the planning steps required.

Ichter et al. [67] proposed a non-uniform sampling procedure where a sampling distribution is learned from demonstrations and subsequently used to bias the sampling process. A conditional variational autoencoder generates samples within regions of the configuration space likely to contain the optimal solution. When combined with a sampling-based path planner, this strategy effectively harnesses the underlying structure of a planning problem while retaining the theoretical guarantees of sampling-based approaches.

Zucker et al. [193] introduced an RL-based adaptive sampling strategy, which involves sampling the discretized workspace and the corresponding joint configuration. Each episode is assigned a reward, and the process is optimized through gradient descent of the reward with respect to the parameters of the workspace sampling probability distribution.

Conversely, [186] focuses on learning a rejection sampling technique

aimed at discarding samples deemed ineffective in addressing the problem at hand, while [65] employs Q-learning for the tree-expansion process, aiming to prevent collisions and reduce expansion into irrelevant areas.

This category of methods has garnered substantial interest from the scientific community, leveraging the impressive capacity of ANN to approximate complex functions, coupled with RL's proficiency in acquiring optimal behavior in dynamic environments. Nevertheless, it is crucial to acknowledge that these methods present certain limitations, primarily related to the training process. They often necessitate significant amounts of data and computational resources for effective training, typically tailored to a specific robot within a defined environment. Furthermore, while these models demonstrate proficiency in the environments they were trained on, their performance may experience a decline when faced with unfamiliar settings featuring entirely unpredictable obstacles.

2.3.3 The graph-based approach

A very common and well-known methodology to solve the path planning problem is represented by the graph-based approach. Basically, it consists of approximating the search space with a graph of discrete states, called *vertices*, connected by *edges*. With this methodology, the problem is dual. Firstly, the continuous search space needs to be discretized to build the graph. Then, the graph must be searched to find the path from the starting vertex to the goal vertex.

Choosing the correct space discretization strongly affects the accuracy of the space representation and the performance of the path planner. Low resolution allows for a faster search but produces a worse path quality. High resolution allows for smoother paths but requires expensive search. Thus, researchers have spent a lot of effort on different type of discretization, including visibility graphs, regular, non uniform and

multi resolution graphs [19, 106, 116].

Formal guarantees of the graph-based algorithms respect to the continuous path planning problem depends on the *a priori* discretization. In particular, an algorithm is said *resolution complete* if it guarantees to find a solution to a given problem at the chosen resolution, if one exists. If it guarantees to find the optimal solution, if one exists, it is said *resolution optimal*. By definition, resolution optimality implies resolution completeness [46].

The most popular algorithms to find a path in the graph are the Dijkstra's algorithm [31] and A* [57]. These algorithms solve single-query planning scenarios and find the optimal solution in an efficient way by ordering the search based on cost. In this way, the optimal solution is found only after all possibly better paths have been considered [46].

Dijkstra's algorithm orders the search based on the *cost-to-come*, which is the cost to reach a specified vertex from start. Basically, the algorithm uses a queue of vertex ordered by the cost-to-come. At each iteration, the lowest-cost vertex is extracted, the costs of the descendants vertices are computed and then they are inserted into the queue. The process continues until the goal is extracted from the queue, that means that all the vertices with lower cost-to-come have been already considered. This process avoids expanding all the vertices with higher cost-to-come that could not provide a better path. Moreover, the algorithm can be adapted for multi-query problems, concluding only when the queue is completely empty instead of just waiting for the goal to be extracted.

A* improves the efficiency of the search ordering the queue based on the *potential solution quality*. Each vertex v is evaluated by the function $f(v) = g(v) + h(v)$, where $g(v)$ is the cost-to-come of v, while $h(v)$ is an heuristic estimate of the *cost-to-go* of v, which is the cost to move from v to the goal on the optimal path. If $h(v)$ never overstimates the real cost-to-go of v, $f(v)$ is an *admissible* estimate of the cost of the optimal

solution from start to goal constrained to pass through v. This admissible estimate allows to exclude those vertices that certainly will not provide a better solution. The resulting search is *optimally efficient* in the number of vertices expanded since any other resolution-optimal algorithm using the same heuristic will expand at least the same number of vertices as A* [57]. Enhancing the precision of the heuristic leads to the expansion of fewer vertices. Heuristic h_1 is considered more informed than heuristic h_2 if, for all vertices v, it holds that $h_1(v) > h_2(v)$. The least informed heuristic corresponds to $h(v) = 0$ for all vertices, which transforms A* into Dijkstra's algorithm.

The notion of *admissible heuristics* holds significant importance in the realm of path planning and will be a recurring theme in this thesis. In essence, a heuristic is considered *admissible* when it consistently provides an estimate that is never higher than the actual cost of transitioning between two points. For instance, consider a graph in a plane where edge costs are defined by the Euclidean distance between vertices. In this context, the Euclidean distance to the goal stands as an admissible heuristic, representing the absolute minimum distance needed to connect two points—a lower-bound for the potential solution's cost.

The illustration of Figure 2.2 depicts the A* search algorithm in action, finding a path from the initial point S to the destination point G. This visualization highlights how the pre-defined grid resolution significantly impacts the resulting solution's cost. Increasing the resolution would indeed yield a shorter and smoother path; however, this enhancement comes at the cost of expanding a greater number of vertices, consequently prolonging the time needed to find the solution.

The primary limitation of both Dijkstra's algorithm and A* lies in their tendency to explore numerous vertices in order to identify the optimal solution, a challenge that becomes particularly pronounced in expansive search spaces, where they lead to increased running time and

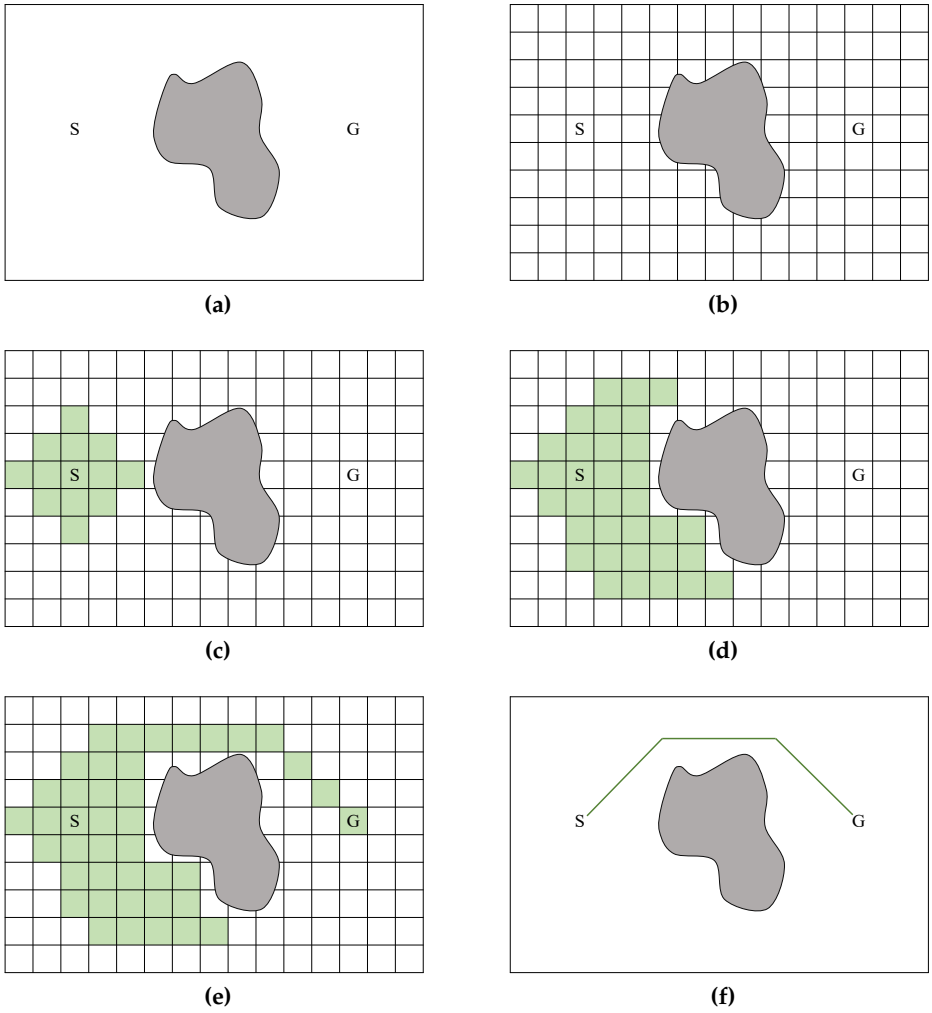


Figure 2.2: An illustration of how A* [57] solves the problem of finding a feasible path from S to G. The continuous search space is discretized into a grid, (a). Starting from S, nodes are expanded ordered by the cost-to-come plus an *admissible* heuristic of the cost-to-go, (b), (c), (d). The optimal solution is found when the goal node is expanded, which means that all the possible less-costly paths have already been considered, (e). The resulting path is the *resolution optimal* one, (f).

memory requirements. Further researches have been done to speed-up the search process, including *near resolution optimal* algorithms, *replanning* or *incremental searches* and *anytime searches*. These approaches aim to accelerate path finding, a crucial aspect not only for swiftly determining the initial solution but also for the potential adaptation or modification of the path in real-time, especially in response to new environmental data.

Near resolution optimal algorithms expedite the search process by loosening the strict optimality criterion of the sought solution. In numerous applications, attaining the absolute optimal path may not be critical; instead, achieving an almost optimal solution (the *near resolution optimal solution*) in a shorter time suffices.

One effective approach is to inflate the heuristic by a factor $\epsilon > 1$, as introduced in Weighted A* [132]. This adjustment biases the search towards states closer to the goal, resulting in a reduced number of expanded vertices needed to find a solution. Importantly, the solution obtained is guaranteed to have a cost no greater than ϵ times that of the resolution-optimal path. It is noteworthy that ϵ can be dynamically adjusted throughout the search process [133]. Additionally, Cohen et al. proposed Lazy Weighted A*, a modification of Weighted A* that evaluates the cost of edges only when absolutely necessary, thereby minimizing time spent in extensive search spaces [25]. Ebendt et al. [35] provide a comprehensive overview and analysis of this strategy, along with its various adaptations.

Multi-Heuristic A* (MHA*) [3] leverages a combination of inadmissible heuristics alongside an admissible one, effectively integrating diverse sources of information into the search process. By doing so, it enables the discovery of solutions that are nearly optimal, with the degree of deviation from the optimal path controlled by a user-defined parameter.

Replanning or *incremental searches* are tailored for graphs that undergo dynamic changes. This makes them especially adept for environments

that evolve over time, where new information about the surroundings may be gathered during navigation, such as the emergence or movement of new obstacles. These algorithms strategically plan a path, considering it unobstructed in areas where environmental details are lacking, and dynamically adjust it in real-time using existing knowledge to swiftly refine the solution. As a result, they adapt the plan without the need to start A* from scratch. While the overall path taken by the robot may not always be the most efficient, each time these algorithms compute a resolution optimal solution from the current robot position to the goal.

Dynamic A* (D*) [153], Focused D* [154], D* Lite [84], and Lifelong Planning A* (LPA*) [85] stand out as widely used algorithms for graph replanning.

Notably, LPA* [85] represents a variant of A* that dynamically updates the graph to maintain resolution optimality in response to changes in edge costs or when new vertices are added or deleted. Essentially, the algorithm identifies and adds *inconsistent* vertices to a queue, which are vertices with a cost-to-come different from the best cost obtained from their neighbors, referred to as the *lookahead value*. Like A*, the search is prioritized based on the potential cost of the solutions. While the initial search of LPA* mirrors that of A*, all subsequent searches are significantly faster. LPA* yields, at a minimum, the search tree that A* constructs. However, it achieves a substantial acceleration over A* by reutilizing the parts of the prior search tree that align with the new search tree. Figure 2.3 illustrates how LPA* updates the graph search following an obstacle's displacement.

D* Lite [84] is a streamlined and more efficient version of the D* algorithm [153], specifically tailored for robot navigation. It can be viewed as an adaptation of LPA* with distinct characteristics. Unlike LPA*, D* Lite reverses the search direction, initiating from the goal and progressing towards the robot's current position. Additionally, it incorporates a

clever strategy to circumvent the need for recalculating the priority of each vertex in the queue after a robot motion, thereby significantly reducing computation time.

Different speed-up techniques can be integrated, as demonstrated in Truncated D* Lite and Truncated LPA* [2], which blend replanning with a nearly optimal search. These methods prematurely halt the propagation of changes, effectively lessening the computational burden while still maintaining a solution that is nearly optimal, with a cost no more than a user-defined parameter worse than the resolution optimal one.

Anytime searches play a vital role in circumventing time constraints in robotics path planning. While A* guarantees a solution only if there is enough time to find the resolution optimal path, this behavior is not always practical in real-world scenarios. It is often preferable to discover the *best-possible* solution within a given time frame. This allows for continuous refinement of the solution during the remaining time or execution. The anytime approach is widely employed in motion planning and will be extensively discussed in this thesis. It proves especially valuable in applications involving obstacle avoidance, where a collision-free path must be computed rapidly. In such cases, prioritizing a safe path over an optimal one is paramount. Therefore, it is more practical to initially compute a lower-quality solution and subsequently improve it over time.

Anytime A* [191] involves repeated executions of A* with progressively less inflated heuristics and Anytime Repairing A* (ARA*) [102] leverages progress from prior iterations for faster execution compared to Anytime A*. Both algorithms ensure suboptimality bounds.

Similarly, Anytime D* (AD*) [101] combines near optimality and incremental searches, allowing it to accommodate dynamic graphs. Additionally, Truncated AD* [2] introduces a strategy for prematurely halting changes propagation.

Numerous other methods have been introduced to enhance the speed

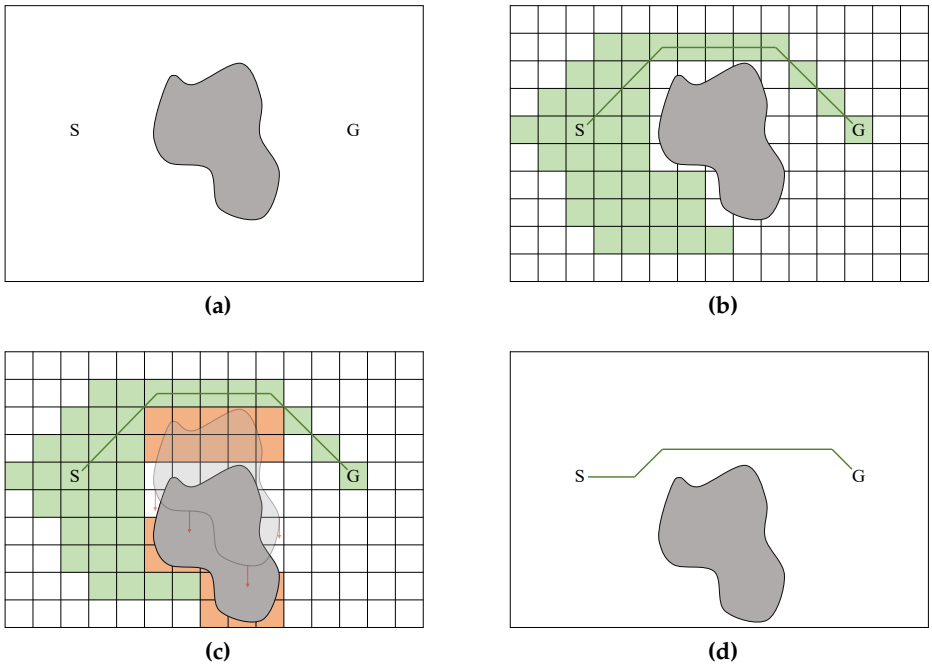


Figure 2.3: An illustration of how LPA* [85] solves the problem of finding a feasible path from S to G in the presence of dynamic obstacles. Initially, the problem is addressed as depicted in Figure 2.2, (a) and (b). As the obstacle shifts its position, the *inconsistent* vertices (orange cells) are inserted into the queue and their cost change is propagated to their respective descendants, (c). The resultant path is determined by updating the search locally, avoiding the need to restart A* from the beginning, (d).

and efficiency of graph-based searches [55, 103, 111]. However, graph-based algorithms have some drawbacks. They require pre-defining the resolution. This is not a trivial task as it heavily depends on the specific nature of the problem. Finding the optimal resolution, one that ensures both high-quality solutions and efficient computation times, can be a non-trivial challenge. Furthermore, it is important to note that the complexity of the search grows exponentially with the increase in the dimensions of the search space, a phenomenon known as the *curse of dimensionality* [6]. Therefore, while graph-based algorithms are a favored choice for simple planning problems, such as those involving mobile robots, they become less suitable when dealing with manipulators with a high number of degrees of freedom. This limitation becomes particularly apparent when aiming for reduced computation times, as in the case of online replanning.

2.3.4 The sampling-based approach

2.3.4.1 Preliminaries

The sampling-based approach to path planning has garnered prominence in recent years due to its inherent advantages, including simplicity, flexibility, scalability with respect to the dimensionality of the search space, and its independence from direct configuration space representation.

Sampling-based planning algorithms have demonstrated their effectiveness in addressing high-DoF planning challenges. However, they traditionally operate under the assumption of static and known environments. Recent advancements in sampling-based planning have extended these techniques to real-time planning in dynamic or uncertain environments, unlocking novel behaviors for high-DoF robotic systems [151].

This approach surpasses the limitations of the graph-based paradigm by obviating the need for a predefined discretization of the search space.

Instead, it constructs a representation in an *anytime* fashion through *random* [4] or *deterministic* [9] sampling. This representation becomes more accurate with increasing computational time and significantly mitigates the *curse of dimensionality*.

In a sampling-based algorithm, the key components include:

- **The Sampler**: It is responsible for generating samples within the search space. Its primary function is to search for a solution within this space. While the standard approach involves uniform random sampling, extensive efforts are continually invested in refining and guiding this process to efficiently discover improved solutions [48, 49, 67, 165].

- **The Local Planner**: This component is responsible for establishing connections between two sampled points. It must operate with speed and computational efficiency, as it is executed repeatedly in a sampling-based algorithm. The visibility region of a node under a specific local planner refers to the set of states in the search space that the local planner can successfully connect to the node. A broader visibility region indicates a more powerful local planner, often entailing higher computational requirements. Thus, finding the right balance between these factors is paramount. In most instances, the local planner aims to connect two samples through a direct line [53]. However, there are exceptions, such as kinodynamic planners, where local connections are determined by the robot's dynamics and are costly to compute [32].

- **The Collision Checker**: It verifies the feasibility of a sample or connection, ensuring it does not result in a collision. This step is vital as it enables the identification of collision-free paths. However, it is typically the most resource-intensive operation in a sampling-based algorithm, accounting for up to 90% of the computational time [36].

Artificial intelligence and parallelization techniques have been employed to accelerate this process [18, 28, 123].

- **The Neighbor Set**: This set comprises nodes in close proximity to the new samples, which the local planner can attempt to connect with. Generally, this subset is defined by nodes within a certain distance or the K closest nodes, reducing the number of connection attempts. This is essential as the probability of successfully establishing a connection significantly diminishes with increasing distance from the new samples.

Formal guarantees of these algorithms in relation to the original continuous feasible or optimal planning problem (Definitions 1 and 2) are often *probabilistic*.

Definition 4: Probabilistic Completeness

A planner is said to be *probabilistically complete* if, as the number of samples tends to infinity, the probability of finding a *feasible* solution, if one exists, approaches unity:

$$\lim_{N_q \to \infty} P\left(\sigma \in \Sigma, \sigma(0) = q_{\text{start}}, \sigma(1) = q_{\text{goal}}\right) = 1 \tag{2.15}$$

where N_q is the number of samples, σ denotes the path found by the planner using those samples, and Σ represents the set encompassing all feasible paths.

Definition 5: Almost-Sure Asymptotic Optimality

A planner is said to be *almost-surely asymptotic optimal* if, as the number of samples tends to infinity, the probability of converging asymptotically to the *optimal* solution, if one exists, approaches unity:

$$\lim_{N_q \to \infty} P\left(c(\sigma) = c(\sigma^*)\right) = 1 \tag{2.16}$$

where N_q is the number of samples, σ denotes the path found by the planner using those samples, σ^* represents the optimal path and $c(\cdot)$ the cost function.

2.3.4.2 The algorithms

The two prevailing sampling-based algorithms in the field are the Probabilistic RoadMap (PRM) [77] and the Rapidly-exploring Random Tree (RRT) [93].

Probabilistic RoadMap (PRM) [77] is a multi-query method that first constructs a representation of the search space and then searches for a path within the created graph, called *roadmap*. For this reason, it does not differ much from graph-based approaches, except in how the roadmap is constructed. To mitigate the *curse of dimensionality*, the roadmap is created by randomly sampling the search space. Each new sample is connected to the roadmap through a *local planner*, typically resulting in a direct connection. This connection is established if it proves collision-free. Note that, within the sampling-base algorithms, the naming *vertex* and *edge* is often replaced with that of *node* and *connection*. As outlined in Algorithm 1, this phase is known as the *learning phase* and is focused on creating an accurate representation of the search space. It is critical in this phase to integrate the inherent connectivity of the search space into the roadmap. Note that, in line 7 of Algorithm 1, the local plan-

Algorithm 1 Probabilistic RoadMap (PRM) - learning phase

Require: Configuration space $\mathcal{C}$, number of samples N_q, local planner, collision checker
Ensure: PRM graph $\mathcal{G}$

```
Initialize empty graph G
for i = 1 to N_q do
    Sample random configuration q in C
    if CollisionChecker → isFree(q) is True then
        Add q as a node to G
        for all n ∈ a subset of G containing nodes neighboring q do
            if LocalPlanner(n, q) is not in collision then
                Add edges (n, q) and (q, n) to G
return G
```

ner attempts to establish connections only with nodes within a distance threshold or with the K nearest nodes. The subsequent phase, known as the *query phase*, involves solving a specific planning problem by connecting the start and goal configurations to the roadmap and searching for the path using a graph-search algorithm (such as Dijkstra's algorithm or A*). The start and goal are connected to the roadmap by the local planner in the same manner as new samples during the learning phase. PRM is probabilistically complete and, with an appropriate connection scheme, almost-surely asymptotically optimal [76]. Figure 2.4 illustrates both the *learning* and *query phases* of the PRM algorithm.

Roadmaps invest time in covering the entire search space during preprocessing, which in turn enables rapid responses to multiple queries once established. However, for single-query tasks or in dynamic settings, creating a roadmap solely for one use is often an unnecessary effort. While extensive research has been conducted to enhance PRM [8, 115, 117], for single-query applications an incremental search for a solution remains the more efficient approach.

Rapidly-exploring Random Tree (RRT) [93] stands as a foundational

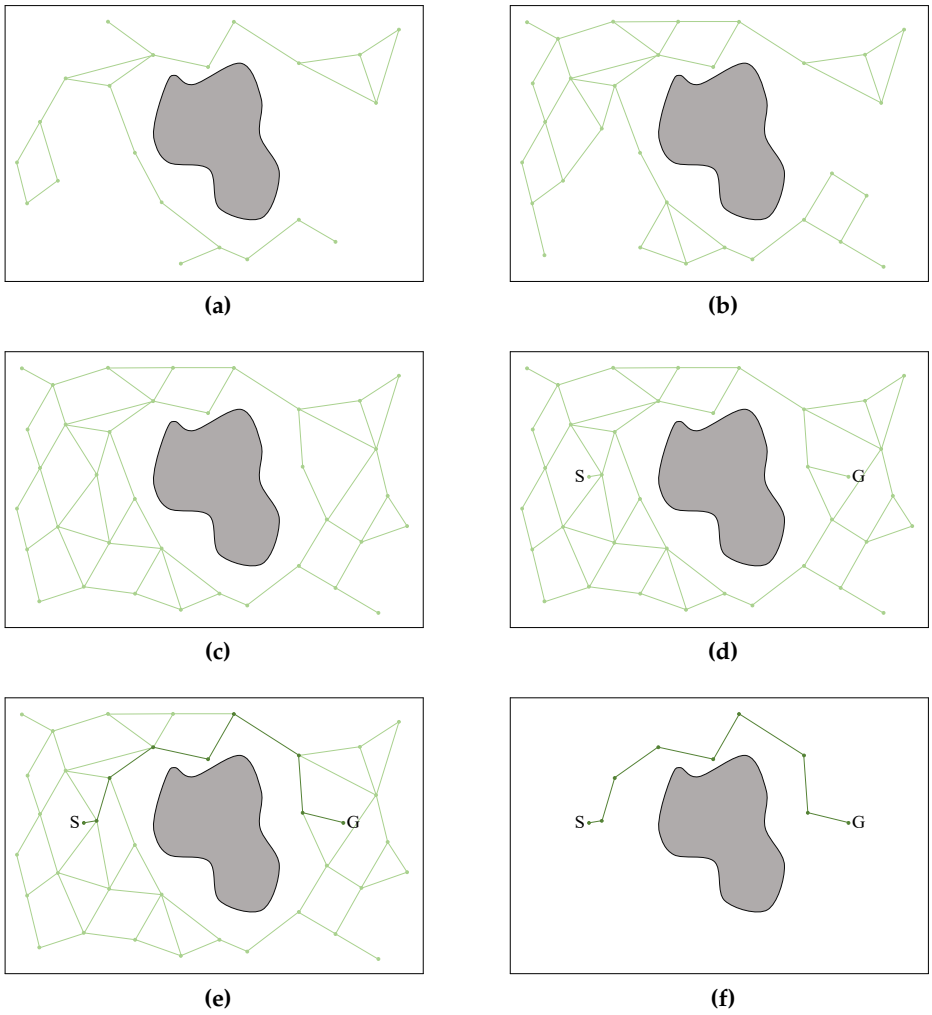


Figure 2.4: PRM [77] finds a feasible path from S to G using a two-phase approach. In the *learning phase*, a *roadmap* (*i.e.,* a graph) is constructed to represent the search space. New samples are connected in the collision-free space using a local planner, (a) and (b). This process continues until a satisfactory space representation is achieved, (c). In the *query phase*, S and G are connected to the closest roadmap nodes, (d), and the best path is determined using a graph-search algorithm (*e.g.,* A*), (e) and (f). The algorithm is probabilistically complete and, with an appropriate connection scheme, almost-surely asymptotically optimal.

Algorithm 2 Rapidly-exploring Random Tree (RRT)

Require: Configuration space $\mathcal{C}$, initial configuration q_{init}, goal configuration q_{goal}, number of iterations N_q, max step size η
Ensure: RRT tree $\mathcal{T}$

```
1:  Initialize tree T with root node q_init
2:  for k = 1 to N_q do
3:      Sample random configuration q_rand in C
4:      Find nearest node q_near ∈ T to q_rand
5:      Extend q_near towards q_rand with a max distance η to create q_new
6:      if (q_near, q_new) is collision free then
7:          Add q_new as a child of q_near in T
8:          if (q_new = q_goal) then
9:              return T
10: return T
```

approach in addressing continuous single-query planning problems. It operates by iteratively constructing a tree within the collision-free space, originating from the initial node (see Algorithm 2). During each iteration, a random sample is drawn (line 3) from the search space. The algorithm then extends the tree toward this sample, effectively steering the exploration towards uncharted areas. Specifically, the nearest node in the tree is selected, and a connection is established with a user-defined maximum length, provided it is collision-free (line 7). Notably, the sampling process is typically biased towards the goal using a user-defined goal-sampling probability, significantly enhancing the chances of finding a solution. When the algorithm successfully links the goal to the tree, it produces a solution to the continuous planning problem by retracing from the goal. Figure 2.5 illustrates the sequential steps undertaken by RRT in its quest for a solution. While RRT exhibits probabilistic completeness, it falls short of achieving almost-sure asymptotic optimality [76].

RRT-Connect [87] enhances the basic RRT approach by growing two

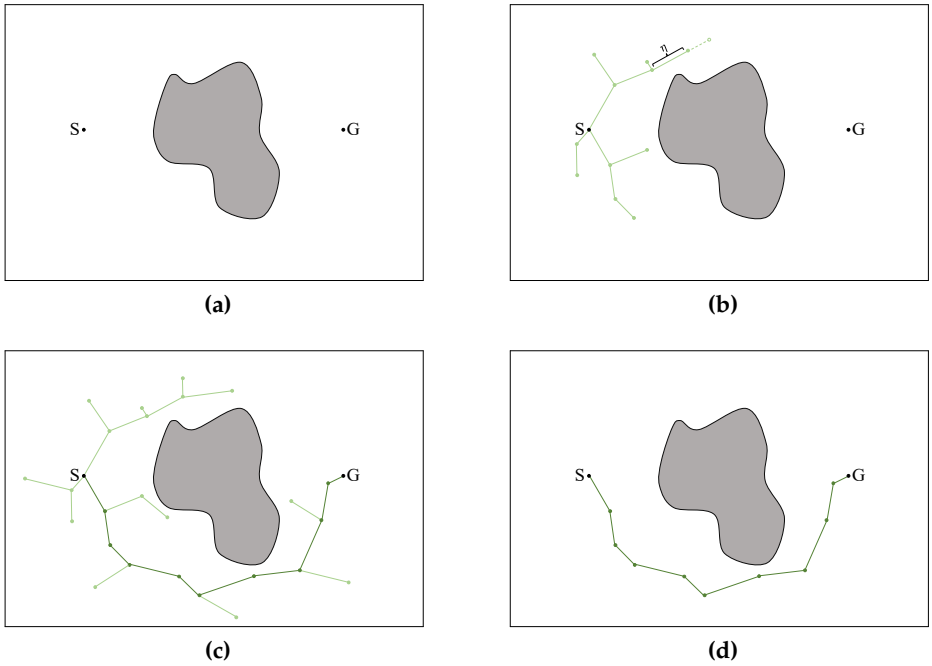


Figure 2.5: An illustration of how RRT [93] solves the problem of finding a feasible path from S to G. The continuous search space, (a), is represented by a sampling-based tree rooted at the start S. The accuracy of the representation improves with additional time in an *anytime* fashion by randomly sampling the search space and growing the tree with nodes and connections with maximum length equal to η in the collision-free space, (b)-(c). The algorithms guarantees to find a feasible solution, if one exists, but not to almost-surely converge to the optimal one, (d).

trees, one starting from the initial node and the other from the goal node. In each iteration, a random sample is drawn, and the two trees take turns trying to extend towards it. Unlike the original RRT algorithm, RRT-Connect extends the tree from the closest node until it reaches the new sample or an obstacle. This dual-tree structure enables faster exploration of the search space, generally leading to quicker solutions compared to standard RRT. Similar to RRT, RRT-Connect is probabilistically complete but not almost-surely asymptotically optimal, thus solutions often require post-processing [107].

While nodes may be optimally connected within the *current tree*, connections may need to be adjusted as the trees continue to grow to maintain optimal node connectivity. RRG and RRT* [76] achieve almost-sure asymptotic optimality by taking into account both incoming and outgoing connections of newly added nodes to enhance the cost-to-come of existing nodes. RRG constructs a graph, whereas RRT* maintains a tree structure. Specifically, what sets RRT* apart from RRT is its incorporation of the *local rewiring* procedure. Instead of directly connecting a new sample to the nearest node, it links to the neighbor that offers a reduction in the cost-to-come. Subsequently, neighbors that stand to benefit from the new node as their parent undergo rewiring. This continuous rewiring process enhances the tree's internal connectivity. Selecting the appropriate neighborhood size is crucial for achieving asymptotic optimality [76]. It is important to emphasize that optimality is theoretically guaranteed with infinite samples; therefore, the rewiring process persists even after discovering an initial feasible solution. However, in practical applications, a maximum limit on iterations or computation time is typically set, at which point the algorithm terminates. Figure 2.6 provides an illustration of how the algorithm almost-surely asymptotically converges to the optimal solution.

The primary limitation of RRT* is its slow convergence to the opti-

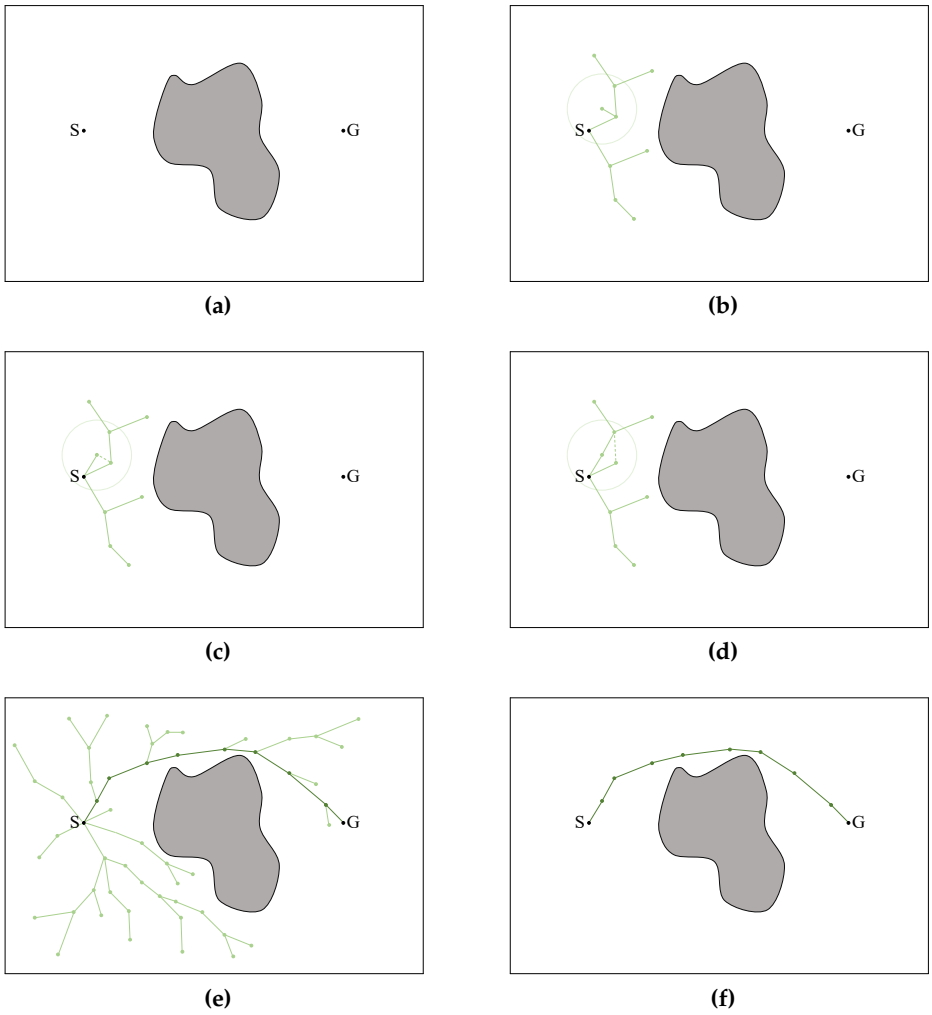


Figure 2.6: An illustration of how RRT* [76] solves the problem of finding a feasible path from S to G. The continuous search space, (a), is represented by a sampling-based tree rooted at the start S, (b). Each new sample is connected to its best neighbour, (c), and it is used to improve connectivity of existing neighbours, (d). The accuracy of the representation improves with additional time in an *anytime* fashion by randomly sampling the search space, (e). The algorithms guarantees to find a feasible solution, if one exists, and to almost-surely converge to the optimal one with increasing number of samples, (f).

mal solution. It proves to be inefficient for single-query problems due to its agnosticism towards the specific planning context. This means it dedicates effort to finding the optimal path to every state in the problem domain, even focusing in areas that may not significantly contribute to the final solution.

In contrast, Informed RRT* [47] addresses this challenge by dynamically shrinking the search space once an initial solution is achieved. Initially, it follows the same procedure as RRT*, but once a preliminary solution is found, it refines the search by excluding areas that do not contribute to further improvement. Consequently, it is less affected by the curse of dimensionality and converges more swiftly towards the optimal solution.

Specifically, define the *omniscient set* as the subset encompassing all states q that could potentially belong to a better solution compared to the current one. Its size diminishes as the solution refines. Denoting f as the function providing the cost of the optimal path from q_{start} to q_{goal}, constrained to pass through point q, the *omniscient set* is formally defined as:

$$\mathcal{Q}_f := \{q \in \mathcal{C}_{\text{free}} \mid f(q) < c\} \tag{2.17}$$

Here, c represents the cost of the current solution. Obtaining precise knowledge of the *omniscient set* necessitates solving the planning problem. Therefore, *informed sets* are used as estimations. An *informed set* is defined by a heuristic $\hat{f}$ approximating the function f:

$$\mathcal{Q}_{\hat{f}} := \{q \in \mathcal{C}_{\text{free}} \mid \hat{f}(q) < c\} \tag{2.18}$$

While an infinite number of *informed sets* exist, particularly valuable are the *admissible informed sets*. An informed set is deemed *admissible* if the heuristic $\hat{f}$ is also admissible, meaning it never overestimates the true

value of the function f:

$$\forall q \in \mathcal{C}, \hat{f}(q) \leq f(q) \tag{2.19}$$

An admissible informed set is a representative overestimate of the omniscient set and contains it completely. The accuracy of the informed set in representing the omniscient set is contingent on the precision of the admissible heuristic $\hat{f}$ in approximating the function f.

For problems centered on minimizing path length, the Euclidean distance serves as an admissible heuristic, as it corresponds to the shortest path length between two points in the search space. This leads to the admissible informed set defined by the following prolate hyperspheroid [47]:

$$\mathcal{I} = \{q \in \mathcal{C}_{\text{free}} \mid ||q - q_{\text{start}}||_2 + ||q_{\text{goal}} - q||_2 < c\} \tag{2.20}$$

By directly sampling from the subset defined by Eq. (2.20), as outlined in [47], it becomes possible to bypass the sampling of states that are guaranteed not to enhance the solution, thereby expediting the optimization process. Figure 2.7 highlights the ability of Informed RRT* to focus the search in a narrow area containing the optimal solution and Figure 2.8 compares the exploration space of Informed RRT* to that of RRT*.

While [47] presents a clear and effective informed set, adapting this method to different cost functions can be challenging (*i.e.*, directly sampling diverse informed sets proves to be a complex task). Additionally, while the proposed approach exhibits strong performance once an initial solution is obtained, if the planner encounters difficulty in finding this solution, the computation time remains high. Enhancing the speed of this aspect has been a subject of interest for numerous researchers [58, 80, 97].

Many efforts are spent to improve the performance of these algorithms in finding a solution even in complex scenarios and in reaching

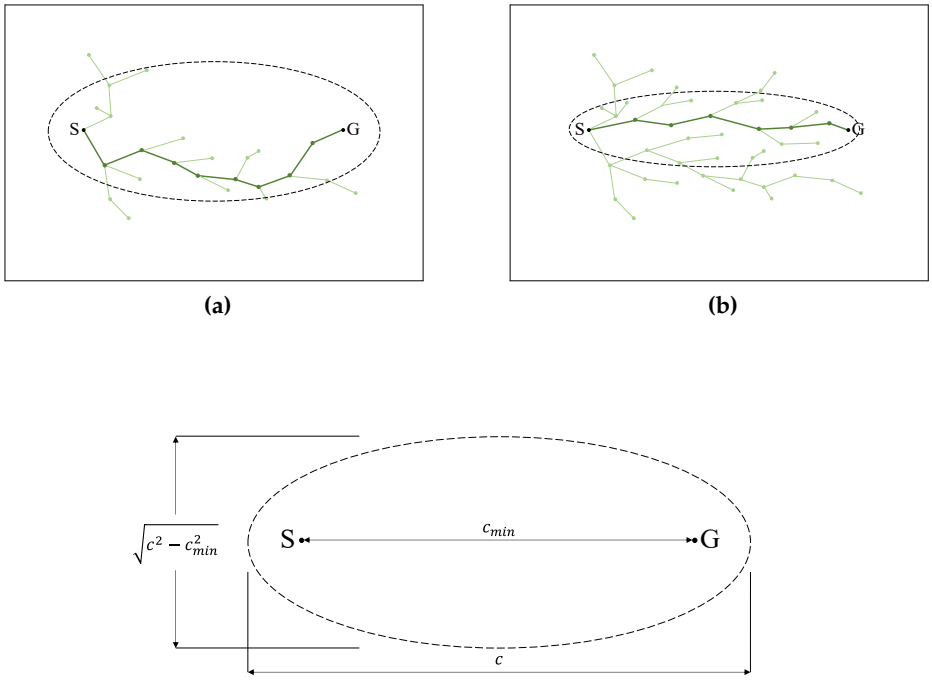


Figure 2.7: After obtaining an initial solution, Informed RRT* [47] restricts the search space within an admissible informed set to speed up the search for a better solution, (a). As the solution improves, the search area narrows, (b). By employing the Euclidean distance as an admissible heuristic, the informed set manifests as a prolate hyperspheroid, equivalent to an ellipse in two-dimensional problems, (c).

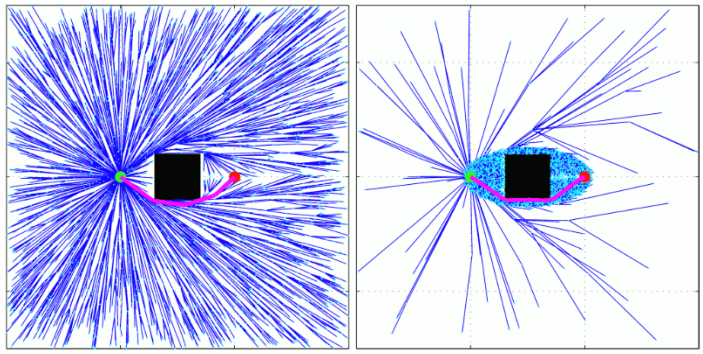


Figure 2.8: Illustration of the search space explored by RRT* [76] (left) and by Informed RRT* [47] (right). Informed RRT* focus on a narrow area and finds the optimal solution faster compared to RRT*. Retrieved from [49].

the optimum quickly. However, in dynamic contexts these algorithms do not provide solution quickly enough to make the robot reactive to obstacle motions. Replanning algorithm usually try to speed up the computation of new solutions by reusing as much as possible efforts spent during the past planning phase.

With PRM, it translates into updating node and connection costs and searching for the optimal path within the updated graph. A popular algorithm derived from PRM is Dynamic Roadmaps (DRM) [95, 96]. The characteristic of this method lies in mapping the roadmap into a discretized representation of the workspace. In practice, in a first phase a roadmap is created in the configuration space $\mathcal{C}$. Differently from other approaches, the roadmap built corresponds to an obstacle-free environments. Then, each node and connection in the roadmap is associated with a cell (*voxel*) in the workspace. These two phases are specific to the robot, but are independent of the environment in which the robot will operate. In the online-planning phase, the planner identifies the voxels occupied by obstacles and invalidates the corresponding nodes and connections from the roadmap, using the encoded mapping. Precise collision check

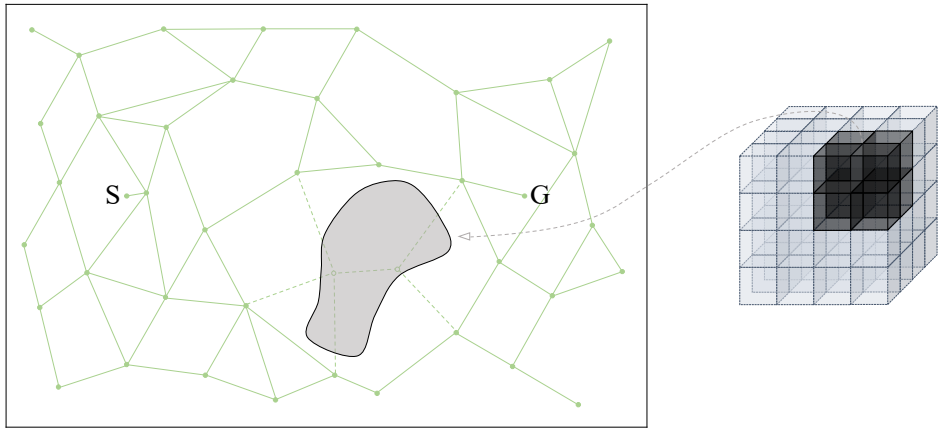


Figure 2.9: DRM [95] involves partitioning the workspace (depicted on the right) into discrete cells (grey boxes) and associating each node and connection of the roadmap (shown on the left) with specific cells. When an obstacle occupies certain cells (black boxes), the relevant nodes and connections are invalidated (dotted connections).

can be done or reference counting could be exploited to speed-up validation [73]. Then a solution is searched with a standard graph-search algorithm without the need of further collision checking. Figure 2.9 depicts the correspondence between the roadmap and the workspace.

DRM has catalyzed various advancements in its implementation. For instance, [88] employs efficient distance metrics and data structures to ascertain neighboring nodes, and utilizes A* for graph traversal rather than the depth and breadth-first searches found in [95, 96]. Furthermore, [104] integrates DRM with Lazy PRM [8], and streamlines the mapping process by focusing solely on nodes. In the event of a cell collision, all corresponding nodes are invalidated, while connections are assumed to be collision-free. The graph is explored using A*, with collision-checking to validate the solution. If a connection encounters a collision it becomes invalid and the search process restarts. This approach enables swifter mapping without compromising online planning times compared

to DRM. The Parallel DRM (PDRM) algorithm [149] harnesses the power of parallel computing to accelerate numerous facets of DRM. This includes tasks like identifying occupied voxels, connecting start and goal to the roadmap and searching the graph. Distance-Aware DRM (DA-DRM) [82] is capable of planning trajectories, even in unknown environments, which are aware of the distance from the obstacles obtained through a depth camera system. Respect to the standard DRM approach, it penalizes nodes and connections associated to voxels close to the ones occupied by obstacles. As a result, the distance is considered as a cost factor during the graph-search process in order to prioritize paths away from obstacles. Cell-PRM (CPRM) [81] employs an anytime approach in constructing its roadmap. It partitions the search space into cells and directs sampling towards cells near the optimal path and that have unconnected components. When the start and goal points belong to the same connected component, a path can be determined. This approach is anytime as sampling can continue to refine the resultant path, and replanning can be executed by resampling the affected cells.

Elastic roadmaps [180] and Reactive Deformation Roadmaps [51] employ control strategies to dynamically adjust the roadmap in response to changes in the environment. In [182], a potential field is utilized to steer nodes away from moving obstacles, although additional collision checks are needed during the query phase.

Lazy SISPRM [98] introduces time as an extra dimension and leverages a human motion prediction system for real-time planning of safe and efficient trajectories. The algorithm plans a path considering the future states of obstacles and disregarding time. It then constructs a Shortcut PRM (SPRM) along this path, linking sequential waypoints. Next, the algorithm introduces time and determines collision-free intervals for each node. Finally, an optimal graph-search is performed in SPRM $\times\, \mathcal{T}$, employing a lazy approach for path validation.

Temporal PRM (T-PRM) [66] augments the nodes in the roadmap with timing information. It specifically calculates, based on predicted obstacle movements, the time intervals in which each node remains free from collisions. These intervals are subsequently factored into the graph-search process, conducted via a modified A* algorithm, to ensure avoidance of intersections with obstacles.

Methods that incorporate the time dimension often rely on an obstacle motion prediction system, which can be prone to inaccuracies. To address this issue, [166] tackles the problem by expanding the size of obstacles as the prediction becomes less reliable. The graph is explored in an anytime fashion using AD* [101], facilitating rapid solution identification with ongoing refinement.

Yet, in highly dynamic environments, obtaining accurate predictions of obstacle motion is often challenging, leading to the adoption of overly cautious motion estimations. This can lead to disconnected components within the roadmap, rendering it unable to generate new solutions or resulting in lower-quality paths.

RRT-like algorithms are traditionally single-query planners that grow a tree from the start configuration and, as such, are problem-specific. This makes them inherently useful for dynamic planning where the start configuration and workspace can rapidly change. Typical replanning strategies aims to exploit as much as possible information and search efforts from previous plans.

Execution Extended RRT (ERRT) [11] searches for a new path to the goal alternating the sampling of new states with the sampling of the *waypoints cache*, namely the set of nodes that constituted the previous path to the goal. The algorithm exploits temporal coherence and works well in scenarios were changes are small and states form previous plans can be re-used. 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. The waypoint cache concept is re-used in a number of other algorithms presented.

Dynamic RRT (DRRT) [40] employs a different approach, reusing the valid branches of the RRT tree. In the event of an obstacle obstructing the current path, the algorithm identifies and prunes the invalid tree branches. If the tree no longer extends to the goal, it undergoes a regrowth process to reestablish a connection. It is worth noting that, akin to D* Lite [84], the tree expands in reverse, originating from the goal and advancing towards the current robot configuration. This obviates the need for constant repositioning of the tree root as the robot moves. Additionally, sampling can be biased towards regions affected by changes. DRRT involves an initial overhead because it necessitates the verification of all tree branches. Furthermore, the tree is reconstructed using RRT, resulting in the new path being suboptimal. Figure 2.10 shows the replanning process.

Anytime Dynamic RRT [42] serves as an anytime version of DRRT. In a nutshell, DRRT comes into play when an obstacle hinders the current path. Following this, Anytime RRT [41] is systematically employed to improve the existing solution. By discarding nodes whose sum of cost-to-come and distance to the goal is worse than the current solution cost, Anytime RRT assembles a sequence of trees that ensure solutions improve the current one. This combination of algorithms enables dynamic replanning with ongoing optimization.

Connell et al. [27] use RRT* to repair the current path when a potential collision is estimated. Firstly, the algorithm adds a node corresponding to the robot's current position. Then, a node on the current path behind the obstacle is chosen as the replanning goal. Using the distance to the replanning goal as a metric, it establishes a sampling area around the obstacle. The nodes inside this area are rewired to make the node corre-

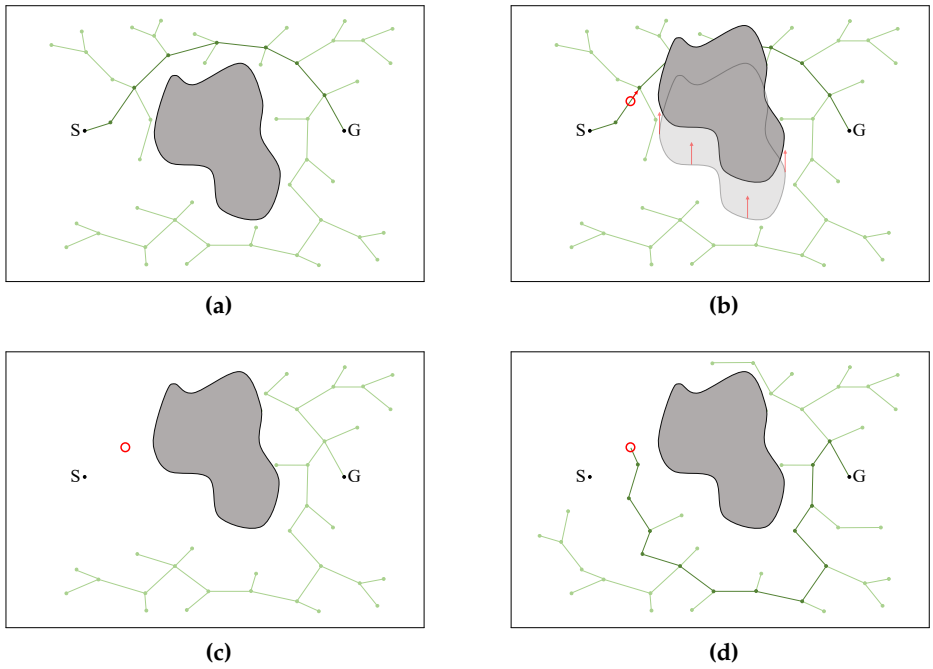


Figure 2.10: This illustration shows the dynamic replanning process in DRRT [40]. Initially, the algorithm generates an RRT, (a). As the robot follows the path, any movement of obstacles can render the path invalid, (b). The red circle signifies the current robot configuration. At this juncture, the algorithm prunes the branches of the tree that have been invalidated by the obstacle, (c). Consequently, the current robot configuration is no longer connected to the goal, prompting DRRT to initiate a regrowth process to establish a new connection and ascertain a fresh valid path, (d). The tree is rooted at the goal G to avoid regrowing the entire tree as the robot moves or new obstacles appear.

sponding to the robot position as their parent node, and new states are sampled to find a path to the goal. A similar methodology is adopted by [188], which incorporates obstacle motion prediction and collision probability assessment.

Reconfigurable Random Forests (RRF) [99] is a multi-query algorithm that 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 to best cover the free space and continuously tries to connect them to each other. Lazy Reconfiguration Forest (LRF) [50] takes a similar approach but validates only the connections belonging to the current solution.

Multipartite RRT (MP-RRT) [194] combines the approaches of RRF, DRRT, and ERRT to mantain a forest of disconnected RRTs and quickly repair the path to the goal. The sampling procedure probabilistically samples the root of disconnected subtrees, the goal or random configurations.

Sun et al. [157] proposed a parallel version of RRT (MPRRT), which continuously runs independent RRTs on each available processor core. The authors demonstrate that MPRRT asymptotically approaches the optimal plan as computational power increases.

Online RRT* (ORRT*) and Online FMT* (OFMT*) [16] adapt to start and end point changes without the need to build an entirely new tree from scratch. They improve the tree by rewiring without adding new nodes. This approach allows adapting to changing environments and cost functions while maintaining constant memory usage.

RRT^x [121] operates with both a graph and a tree, employing a rewire cascade technique to efficiently update the tree when new obstacles appear or shift. In static environments, it has been demonstrated to achieve asymptotic optimality with a faster convergence rate compared to RRT*.

Horizon-based Lazy RRT (HLRRT) [20] verifies path feasibility within

a user-defined time-horizon, eliminating all unfeasible nodes and their subsequent successors to maintain a single pruned tree. It subsequently regrows and rewires this tree by using a sampling distribution based on Gaussian Mixture Models.

Efficient Bias-goal Factor RRT (EBG-RRT) [183] preserves the portion of the path beyond the obstacle and endeavors to re-establish a connection to this path by employing an modified waypoint cache.

Broadly speaking, RRT-like methods have experienced significant growth in dynamic environment planning in recent years [17, 26, 135, 150] and the most common approach typically entails pruning the invalid branches of the tree followed by regrowing and/or applying rewiring techniques. Nevertheless, many of these techniques still struggle in scenarios characterized by high-dimensional search spaces with many sizeable obstacles.

2.4 Safety requirements for HRC

The factory of the future envisions seamless collaboration between humans and robots as a means to enhance industrial productivity and improve workers' conditions [143]. Although there is still no standard and unambiguous taxonomy for the various modes of Human-Robot Collaboration (HRC) in the state of the art [170], a common distinction includes the following types of interactions [120]:

***Coexistence*:** This scenario entails the human and robot concurrently executing separate tasks in different areas without the need for physical barriers. For instance, a robot might handle heavy loads while the operator assesses quality standards.

***Synchronization*:** This method involves the human and robot sharing the same workspace, albeit at different times and in a sequential

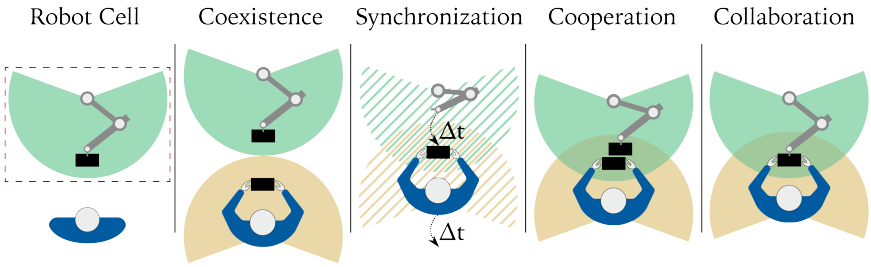


Figure 2.11: Different HRC modes. Inspired by [174].

manner. They perform specific tasks in a coordinated sequence to achieve a common objective. For instance, a human may load raw materials into a machine, followed by the robot executing the machining process. Synchronization is crucial for successful execution.

***Cooperation*:** In this scenario, robots and humans share a common workspace while pursuing distinct tasks and objectives. They have equal access to resources for gathering information about the work and environment. However, there is no direct interaction between them. Although their workspaces may intersect, they do not hinder each other's progress as they strive towards a shared goal. An illustrative example is a human and a robot working in the same robotic cell, each focused on different objects.

***Collaboration*:** In this mode, humans and robots work in the same workspace, towards the same objective, and simultaneously. They operate in close proximity, with controlled contact facilitated by advanced sensing technologies. Actions taken by one entity have immediate effects on the other. Consequently, their connection can be physical, involving forces and torques, or non-contact, with information exchanged through speech, gestures, intention recognition

Table 2.1: Different level of interaction in HRC.

	Coexistence	Synchronization
Work Task		✓
Direct Contact		✓
Simultaneous process	✓	
Workspace		✓
Sequential process		✓
	Cooperation	**Collaboration**
Work Task		✓
Direct Contact		✓
Simultaneous process	✓	✓
Workspace	✓	✓
Sequential process	✓	

and eye tracking.

These different modes are summarized in the Table 2.1 and illustrated in Figure 2.11.

Collaborative applications must comply with safety standards to mitigate potential health risks for workers [170]. This is particularly crucial as the robot is no longer confined to an enclosed space. Safety requirements for collaborative applications are defined by ISO/TS15066 (*Robots and robotic devices – Collaborative robots* [70]). The document outlines four collaborative operations, namely *Safety Rated Monitored Stop*, *Hand Guiding*, *Speed and Separation Monitoring* and *Power and Force Limiting*. Specifically, the last two are of particular interest for this thesis.

Speed and Separation Monitoring (SSM) focuses on maintaining a minimum separation distance between the human and the robot to ensure that the robot can stop before colliding with the person. Specifically, the minimum human-robot distance, denoted as S, should always satisfy the condition $S \geq S_p$. S_p represents the protective separation distance. We can derive the maximum safe velocity v_{max} of the robot towards the hu-

man based on the current separation distance S [12]:

$$v_{max} = \sqrt{v_h^2 + (a_s T_r)^2 - 2a_s(C - S)} - a_s T_r - v_h \tag{2.21}$$

In Equation (2.21), v_h denotes the human velocity towards the robot, a_s represents the maximum Cartesian deceleration of the robot towards the human, T_r is the reaction time of the system, and C a parameter accounting for the uncertainty of the perception system.

Power and Force Limiting (PFL) allows the robot to come into contact with a human with non-zero speed, as long as the kinetic energy transferred to the human does not exceed a certain threshold $E \leq E_{max}$, where E and E_{max} are the kinetic energy transferred to the human and the maximum energy that can be transferred without harming the operator, respectively. E_{max} can be translated into a limit on the maximum robot speed:

$$v_{max} = \frac{F_{max}}{\sqrt{k}} \sqrt{m_r^{-1} + m_h^{-1}} \tag{2.22}$$

Here, F_{max} and k represent the maximum contact force and spring constant for a specific body region; m_r and m_h are the effective mass of the robot and the effective mass of the human body region, respectively [56, 70].

Their implementation typically results in robot speed limitations and halts when the human and the robot are in proximity. For this reason, novel methods must be built upon the safety module, enabling efficient and secure robot control. Researchers have investigated this problem at various levels, proposing adaptive task planners [37, 146], motion planners [33, 38, 122], and controllers [29, 63, 134].

Motion planning is a critical aspect of this endeavor, with researchers striving to formulate trajectories that account for the operator's presence and minimize the activation of the safety module. It is possible to re-

duce the idle times caused by the safety intervention by optimizing the robot movements in such a way as to minimize the safety speed limitations. This optimization can be carried out offline, proactively planning trajectories that minimize the need for safety module interventions based on a model estimating the human's occupancy map during the robot's trajectory execution. However, these methods are susceptible to unforeseen movements of the operator [38, 90]. In contrast, reactive approaches adapt the robot's trajectory in real-time based on the current situation [122, 185]. This alternative will be explored in detail in Chapter 5.

2.5 Discussion and thesis contribution

Path planning is a fundamental field of study in autonomous robotics, encompassing four primary approaches: optimization-based, learning-based, graph-based, and sampling-based methods. While conventional path planning typically assumes a static environment, real-world situations are dynamic, necessitating robots to dynamically adjust their plans to steer clear of collisions and reduce disruptions. Planning from scratch whenever an obstacle obstructs the current path is typically inefficient and time-consuming. Therefore, replanning algorithms must prioritize speed and effectiveness in providing viable solutions.

Optimization-based methods provide the flexibility to integrate new constraints, but can be computationally intensive and may not guarantee optimal solutions. Potential field methods, a subset of this category, have gained traction for handling local trajectory adjustments, yet they may encounter challenges in finding global optima due to local minima issues. Additionally, potential fields calculations often consider limited key-points on the robot and obstacles, potentially compromising responsiveness with an increase in key-point density.

Graph-based methods involve discretizing the search space and then

conducting a graph search. While efforts have been made to update the search rather than starting anew when an obstacle moves, these methods face limitations in higher-dimensional search spaces.

Learning-based approaches, leveraging advancements in artificial intelligence, have gained popularity. Reinforcement learning, in particular, offers a efficient methodology for teaching robots reactive strategies in the presence of moving obstacles. However, these methods may not be suited for completly unknown environments with unpredictable obstacles. Neural networks, on the other hand, often play a supportive role in sampling-based algorithms.

Among the various approaches, the sampling-based method stands out for its simplicity, adaptability, scalability with search space size, and lack of necessity for a direct representation of the search space. PRM and RRT are prominent techniques within this category. PRM-based methods are largely multi-query, efficiently updating roadmap costs and path searches in the modified graph. However, sizable roadmaps are required to approximate the search space and generate high-quality paths. Small roadmaps may result in disconnected components when new obstacles arise, rendering a solution unattainable.

A promising direction lies in adapting RRT to dynamic environments, a field that has witnessed substantial growth. These approaches emphasize the reuse of information from prior planning efforts. This often involves biasing sampling towards specific areas, trimming invalid sections of the tree, and regrowing to reestablish a connection to the goal, or updating the existing tree or graph through a rewiring process. Despite these advances, challenges persist, especially in scenarios involving robots with high degrees of freedom and obstacles that invalidate significant portions of the tree/graph.

This thesis presents a novel approach that leverages the potential of sampling-based methods while avoiding the need for roadmaps complex

to maintain or basic tree structures that substantial or numerous obstacles can significantly invalidate. The primary goal of this thesis is to present a reactive path replanning algorithm capable of swiftly adapting the robot's path to accommodate new obstacles. Subsequently, the algorithm will undergo modifications to incorporate human-aware features, enhancing its efficiency for human-robot collaboration applications. In particular:

- Chapter 3 introduces a motion replanning architecture. As outlined in Chapter 1, motion replanning involves overseeing trajectory execution while simultaneously conducting path replanning and collision checks on the traversed path. The proposed architecture comprises three primary threads, each assigned distinct tasks: high-rate trajectory interpolation and command transmission to the robot's controller, execution of the replanning algorithm, and collision checking of the traversed path with updated environmental data. Notably, this architecture offloads the collision checking responsibility from the replanning algorithm for both the current path and the set of precomputed paths that is used by the replanning algorithm of Chapter 4. This optimization allows the replanning algorithm to save time by circumventing these computationally expensive collision checks during the calculation of a new solution.

- Chapter 4 introduces a novel **M**ulti-p**A**th **R**eplanning **S**trategy (MARS) tailored for robots with numerous degrees of freedom. This algorithm leverages a set of pre-computed paths to the same goal, significantly expediting the online replanning process. The underlying idea is that within the search space, multiple viable paths exist and can be harnessed to swiftly redirect the robot's path when the current one becomes infeasible. By avoiding the need to search for solutions directly connected to the goal, MARS efficiently connects the current path to nodes on other available paths,

especially those closer than the goal, streamlining the search. This is achieved through techniques like subtree reuse, lazy collision checks, and informed sampling, culminating in a quicker and more efficient solution discovery process. Additionally, MARS continuously refines the solution in an anytime fashion. The algorithm has been validated to be probabilistic complete and can approach asymptotic optimality when coupled with an optimal planner such as RRT*. Unlike PRM-based approaches, MARS does not require the maintenance of a complex roadmap, but only a few paths to the same goal. In contrast to DRRT-type replanners, it does not rely on a single tree which can be largely invalidated by obstacles. Instead, it endeavors to connect to already available (and closer) paths to exploit their subpaths to the goal. This approach also distinguishes itself from RRF-like methods. Indeed, in RRF-like approaches, multiple disconnected subtrees are linked together. However, it is important to note that the connection of two subtrees does not guarantee a solution to the goal. In contrast, with MARS, once the current path is connected to an available one a solution is assuredly obtained. The algorithm then proceeds to seek even more refined solutions. An experimental campaign compares the performance of MARS with several sampling-based replanning algorithms. Notably, MARS demonstrates a superior capacity in avoiding collisions and adhering to shorter paths. This distinction becomes even more pronounced as the size of the search space expands. This further affirms the effectiveness of employing precomputed paths to expedite the online search for a solution.

- Chapter 5 extends the work presented in Chapters 3 and 4 to enhance performance in the context of human-robot collaboration. In collaborative robotics environments, safety systems oversee the robot's behavior and intervene to ensure the operator's well-being.

However, this often leads to reduced robot speed or even halts, resulting in inefficient collaboration and idle time. To address this, the replanning algorithm is modified to generate new paths as the operator moves, minimizing the need for safety system interventions and thus enhancing collaboration efficiency. This enhancement is embodied in **H**uman-**A**ware **M**ulti-p**A**th **R**eplanning **S**trategy (MARSHA), which combines MARS with a safety-aware cost function to guide the search towards safety-compliant solutions. As this cost function is more computationally demanding than the Euclidean distance metric used in MARS, the original algorithm undergoes modifications to effectively handle this increased computational load. Extensive simulations and real-world experiments were conducted to assess the performance of MARSHA in comparison to standard approaches commonly employed in the context of human-robot collaboration.

- Chapter 6 presents OpenMORE, an open-source sampling-based path replanning library designed to tackle the challenge of dynamic obstacles in real-world environments. Unlike traditional path planning algorithms that assume static obstacles, OpenMORE enables the robot to adapt its path in real-time without halting, ensuring effectiveness in changing conditions. While established sampling-based path planning libraries exist, the same level of development has not been seen in the field of replanning until now. OpenMORE leverages the framework outlined in Chapter 3, providing developers with a pre-built architecture and facilitating the implementation of new path replanning algorithms. Additionally, it serves as a valuable tool for users seeking to easily solve motion replanning problems, incorporating state-of-the-art sampling-based algorithms.

CHAPTER 3

Path replanning framework

This chapter introduces an architecture able to manage the replanning process during the execution of the robot's trajectory. Initially, it will explain why such a framework is necessary in dynamic applications, and why it is not required in structured contexts where standard path planning suffices. Subsequently, the chapter will clarify the composition of the architecture and outline the functions of its various components.
The chapter is mainly based on the materials published in [163].

3.1 The need for an architecture

Conventional path planning algorithms usually assume a static environment. They determine a feasible path based on the robot's initial and goal configurations, or a set of potential goal configurations, and a model describing the obstacles. Subsequently, the path is temporally parameterised to define a trajectory for the robot to follow. Notably, certain algorithms directly generate a trajectory as output, bypassing the need for an additional time parametrisation step [72, 142]. Figure 3.1 shows the

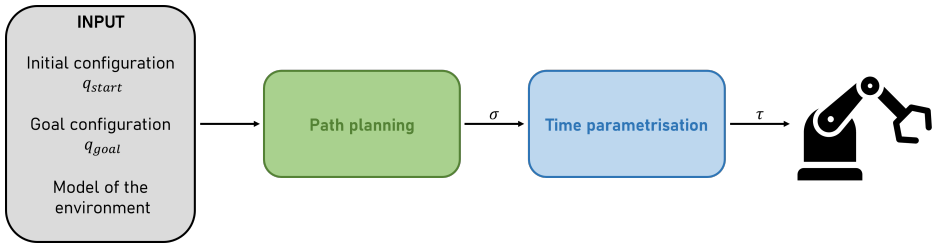


Figure 3.1: The conventional pipeline for path planning algorithms begins with the input of initial and goal configurations, along with a model of the environment. An algorithm like RRT* [76] is employed to calculate a feasible path σ. Subsequently, this path is temporally parameterised to generate a trajectory denoted as τ. This trajectory is then dispatched to the robot's controller for execution.

standard pipeline deployed for path planning algorithms.

The assumption of an unchanging environment proves highly effective in controlled environments, such as those commonly encountered in industrial settings with enclosed workspaces. In these contexts, once a trajectory is calculated, it can be executed without concern for it becoming invalid, as the environment remains constant. If an operator needs to access the robot's workspace for inspection or maintenance, the robot is temporarily halted. It is noteworthy that in these applications, human interventions are typically minimized to maintain the robot's high productivity.

However, this approach falters when faced with dynamic and unpredictable surroundings. As industry evolves towards greater collaboration between humans and robots, the need to remove physical barriers has become imperative. This introduces the challenge of accommodating an environment that is no longer static. For instance, the robot's initially calculated path may need adjustments due to unforeseen changes during the robot's motion. This necessitates the robot to adapt and replan its path on-the-fly.

The stop-and-replan method, where the robot halts and recalculates

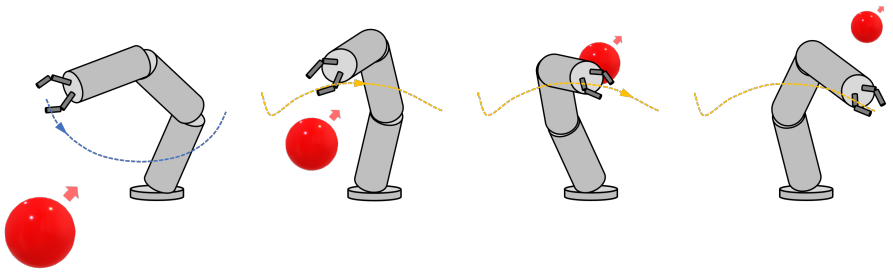


Figure 3.2: Visualisation of a robot manipulator dynamically adjusting its trajectory to avoid a moving obstacle. The initial path (depicted in blue) is modified during the motion, resulting in the generation of a collision-free path (the one in yellow) which the robot begins to follow without interruption.

the path from scratch upon detecting an environmental change, is a rudimentary strategy. While suitable for situations where replanning is infrequent, it proves inefficient when repeated replanning is required. Starting a planning from scratch consumes valuable time, rendering the robot temporarily idle. In a human-robot collaboration scenario, where the operator may frequently obstruct the robot's path, the robot must respond swiftly, possibly without halting, to maintain optimal collaboration efficiency. Achieving this demands not only advanced replanning techniques capable of adapting to environmental shifts in a very short time (few hundreds of milliseconds), but also an architecture to oversee the replanning process concurrently with the execution of the robot's original trajectory. The architecture must execute the replanning process *while the robot is moving*, and must facilitate a *seamless transition* to the new trajectory once identified, *possibly without halting*, as depicted in Figure 3.2.

The combination of a path replanning algorithm and this kind of architecture facilitates path modifications during motion, ensuring a seamless transition towards the newly calculated trajectories. This enables the robot to maintain continuous and agile movement, even in the face of re-

peated obstructions by the operator. This architectural requirement represents a paradigm shift from conventional path planning, which, aside from safety systems that monitor the robot and potentially regulate its speed for operator safety (refer to Section 2.4), lacks real-time supervision during execution.

For the effective implementation of an online path replanning algorithm, the following three tasks must be considered:

1. **Execution of the current trajectory**: The robot's trajectory needs to be followed, continuously sending new commands to the robot's controller at high rate.

2. **Management of the replanning process**: Concurrently, the replanning algorithm must continuously run to identify new collision-free solutions.

3. **Collision checks along the traversed path**: Using up-to-date environmental information, the current path must be continuously monitored for potential collisions. Should a collision be detected, the replanning algorithms must swiftly identify an alternative feasible path.

By ensuring the continuous and simultaneous management of these three aspects, the robot will be able to move and modify its trajectory online effectively in the face of environmental changes, which is a crucial task in modern dynamic workspaces.

3.2 The replanning architecture

The proposed architecture, outlined in Algorithm 3, effectively manages the robot's trajectory execution while concurrently executing the path replanning algorithm. Notably, it is adaptable to various replanning algorithms. It also leverages real-time obstacle information for collision

Algorithm 3 The replanning architecture

Thread *Trajectory execution*

$t \leftarrow t + \Delta t$
state $\leftarrow$ sampleTrajectory(τ, t)
sendToController(state)

Thread *Collision check*

$\sigma_{\text{subpath}} \leftarrow \sigma_{\text{current}}[q_{\text{current}}, q_{\text{goal}}]$
$\mathcal{P} \leftarrow \mathcal{S} \cup \sigma_{\text{subpath}}$
for $\sigma_j \in \mathcal{P}$ **do**
  free, $q_{\text{before}}, q_{\text{after}} \leftarrow$ checkCollisions(σ_j)
  if free **then**
    $c_{\sigma_j} \leftarrow \left\|\sigma_j\right\|$
  else
    $c_{\sigma_j} \leftarrow +\infty$

Thread *Path replanning*

$q_{\text{current}} \leftarrow$ projectOnPath(state, σ_{current})
$\sigma_{\text{replanned}}$, solved $\leftarrow$ replan$(\sigma_{\text{current}}, q_{\text{current}}, \mathcal{S}, \text{max_time})$
if solved **then**
  $\sigma_{\text{current}} \leftarrow \sigma_{\text{replanned}}$
  $\tau \leftarrow$ computeTrajectory$(\sigma_{\text{current}})$
else if distance from obstacle $\leq$ minimum allowed distance **then**
  sendRobotStop()

detection along the robot's current path, enabling the replanning algorithm to generate a new solution based on the latest environmental data. Furthermore, this approach offers the advantage of relieving the collision check of the current path from the replanner, consequently reducing its computational burden.

Specifically, the architecture comprises three threads operating in parallel, each dedicated to one of the three essential tasks for motion replanning presented in Section 3.1:

Trajectory Execution Thread: This thread is dedicated to micro interpolating the robot's current trajectory, leading to the determination of the subsequent robot state (line 2 of Algorithm 3). This state, represented as a tuple $\langle q, \dot{q}, \ddot{q} \rangle$ denoting joint configuration, velocity, and acceleration respectively, is then transmitted to the robot's controller (line 3). The component of the state employed for moving the robot depends on the type of command employed by the controller, such as position or velocity control. Since this thread directly interfaces with the robot's controller and oversees trajectory execution, it operates at the highest frequency among all threads (*e.g.*, 1 kHz).

Collision Checking Thread: The primary focus of this thread is to check the current path σ_{current} for potential collisions. It specifically considers σ_{subpath}, which constitutes the portion of σ_{current} from the current robot configuration q_{current} to the goal (refer to line 4). At line 5, a set of $\mathcal{P}$ paths is considered, comprising both σ_{subpath} and another set of available paths $\mathcal{S}$. At this stage, $\mathcal{S}$ may be considered empty, as its relevance will be clarified in Chapter 4 in connection with the proposed replanning algorithm. Collision checking is specifically carried out at line 7. Additionally, this thread identifies q_{before} and q_{after}, which represent the nodes preceding and succeeding the collision on the current path. These values will also be utilized in Chapter 4.

If the path is found to be free of collisions, its cost is set equal to its length, computed using the Euclidean norm (line 9). Otherwise, its cost is set to infinity, as in line 11.

The collision check in this thread relies on up-to-date environmental information provided by a sensor. As a result, its operational frequency is determined by the rate at which the sensor provides environmental data (*e.g.*, 30 Hz for a standard RGB-D camera).

Path Replanning Thread: The core function of this thread is to trigger the replanning algorithm for the generation of a new path (see line 13). It's not bound to any specific replanning algorithm. Depending on the chosen replanning approach, it may be activated either when an obstacle obstructs the current solution or even when the current solution is unobstructed but a potentially superior one can be found.

Before this, a procedure projects the robot state onto σ_{current} to derive q_{current}. This step is essential because time parameterisation algorithms may introduce slight deviations from the path computed by the path planner (e.g., due to blending radii), whereas the replanning algorithm strictly operates on nodes belonging to the path or tree.

If the replanner discovers a new path, it is assigned to σ_{current}. Then, a time parameterisation is computed for σ_{current} to obtain a new trajectory τ (line 16). This newly computed trajectory will be employed by the *Trajectory Execution Thread* to control the robot's motion.

The duration of a single cycle for this thread hinges primarily on the maximum computational time allotted to the replanner. This time can be adjusted based on specific circumstances. For instance, in cases of path obstruction, a shorter time frame may be designated to achieve a quicker, though potentially less refined, solution. Con-

versely, in scenarios where the path is unobstructed, a more generous computation time can be granted, affording the solver greater leeway to refine the current solution. As a general guideline, in a human-robot collaboration setting, replanning should ideally conclude within 200 milliseconds to emulate the reaction time of a human [151].

It is worth noting that the replanning algorithm receives updates on the current path cost from the *Collision Checking Thread*. This proactive data provision alleviates the need to recalculate the cost of the current path, thereby reducing the computational burden of the algorithm.

In the event that the replanner fails and the robot's proximity to an obstacle becomes critical, a stop signal is activated, and a contingency plan must be implemented.

Hence, this architecture boasts a modular design, with each thread assigned a specific task. This modularity facilitates flexibility, enabling replacement or modification of individual modules to suit application-specific requirements.

3.3 Summary

This chapter introduced an innovative architecture tailored for seamless trajectory execution and concurrent path replanning in dynamic environments. Unlike static environments, new obstacles can emerge and invalidate the initially calculated trajectory in a real context. It is, therefore, necessary for the robot to react quickly to these changes. Thus, a robust architecture at the motion planning level becomes imperative-one that adeptly handles replanning while the robot is in motion, ensuring a seamless transition to a new trajectory when faced with emerging obstacles. The proposed framework manages three essential tasks: executing

the current trajectory, checking for collisions along the current path based on updated environmental information, and replanning the robot's path. Notably, this architecture offers modularity and adaptability while unburdening the replanning algorithm from the onus of collision checking along the current way.

CHAPTER 4

Path replanning algorithm

This Chapter presents 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 just a few hundred milliseconds when an obstacle obstructs the robot's path. The algorithm expedites the search process through the use of informed sampling, constructs a directed graph to recycle results from prior replanning iterations, and continually refines the current solution in an anytime fashion, ensuring the robot remains highly responsive to changes in its surroundings.

In the latter part of the chapter, MARS is compared against state-of-the-art sampling-based path-replanning algorithms through extensive simulations in complex, high-dimensional scenarios. The results demonstrate the ability of MARS to provide better success in avoiding obstacles and higher-quality solutions.

The chapter is based on the work published in [163].

4.1 Introduction

Moving a robot in the real world poses several challenges, including adapting the robot's motion to a dynamic and ever-changing environment. This concern has gained paramount importance in recent years owing to the proliferation of mobile robotics [61], social robotics [118], and the integration of human-robot collaboration in industrial setups [113].

Typically, path planners chart a path from an initial position to a designated goal position, taking into account static obstacles. However, unstructured environments may feature moving and unforeseeable 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 path to avoid collisions and reach the goal safely.

As presented in Chapter 2, four principal categories of approaches emerge to tackle the replanning problem: the optimization-based, the learning-based, the graph-based, and the sampling-based approach. In particular, the sampling-based method shines for its simplicity, adaptability, scalability with search space dimensions, and its lack of dependency on a direct representation of the search space. Notably, PRM and RRT emerge as leading techniques in this category. PRM-based methods grapple with dynamic obstacles by detecting the roadmap segments affected by moving obstacles via a mapping between the configuration space and the workspace, subsequently reinitiating the search for a new feasible solution [73, 95, 96, 104]. Nonetheless, sizable roadmaps are required to approximate the search space and generate high-quality paths. Small roadmaps may lead to disjointed components upon the emergence of new obstacles, rendering a solution unattainable. Another promising solution lies in adapting RRT to dynamic environments. Many existing algorithms typically involve a computationally demanding phase for tree

pruning, followed by a reconstruction step [40, 42, 99, 194]. Alternatively, some approaches focus on updating the existing tree or graph through a rewiring process [27, 121]. Additionally, biasing the sampling towards specific areas proves to be a common strategy [11, 184].

However, formulating an algorithm capable of rapidly replanning the robot's path in complex scenarios, such as high-dimensional search spaces with many obstacles, remains an ongoing challenge.

4.1.1 Contributions

This thesis suggests a novel method to address the shortcomings of current algorithms. Rather than depending on a roadmap complex to maintain or a basic tree structure that can be easily disrupted by new obstacles, this approach uses a set of pre-computed paths to find and progressively refine new solutions. The outcome is a **M**ulti-p**A**th **R**eplanning **S**trategy (MARS), able to quickly provide a suitable solution in complex, high-dimensional search spaces.

At its core, MARS involves the construction of a graph that connects paths to a common goal, allowing the exploitation of a set of paths precalculated. When an obstacle blocks the current path, MARS seeks out an alternative by endeavoring to connect the current path to the other available ones. The algorithm employs a heuristic to ascertain which nodes of the other available paths to connect with. If a connection is found, it can search for better solutions during the remaining time.

Exploiting a set of pre-computed paths in a replanning algorithm is a somewhat novel concept in the literature. The few state-of-the-art approaches that exploit a population of trajectories for replanning use evolutionary computation [169], or Gaussian processes and factor graphs [64, 86]. Instead, our multi-path strategy connects the current path before the obstacle to a node closer than the goal and then uses the available subpath from that node to the goal. This approach serves to diminish

the complexity of the search. Informed sampling [47], subtree reuse, and a directed graph contribute to reduced computations and an augmented convergence rate of the solution, even within complex, high-dimensional scenarios.

A video of the algorithm is also available [1].

4.2 Multi-pAth Replanning Strategy

4.2.1 Preliminaries

This section provides an introduction to key concepts and notation that will be employed in the chapter.

We denote the sequence of $\mathcal{M}$ waypoints (nodes) constituting a path σ as $w_\sigma = (q_1 \dots q_\mathcal{M})$. $\sigma[q_j, q_k]$ is the subpath of σ commencing at node $q_j \in w_\sigma$ and concluding at node $q_k \in w_\sigma$. Furthermore, $\sigma_i \cup \sigma_j : \Omega \times \Omega \to \Omega$ represents a function that concatenates σ_j with σ_i, subject to the requirement that the final node of σ_i matches the initial node of σ_j.

We consider two types of data structures embedded in the configuration space $\mathcal{C}$: a graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ and a tree $\mathcal{T} := (\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T})$ (Figure 4.1). $\mathcal{V}$ and $\mathcal{E}$ are the sets of nodes and connections (or edges) of $\mathcal{G}$, respectively, while $\mathcal{V}_\mathcal{T} \subset \mathcal{V}$ and $\mathcal{E}_\mathcal{T} \subset \mathcal{E}$ are the sets of nodes and connections of $\mathcal{T}$. Every node of $\mathcal{T}$ possesses a single parent and can have multiple children except the tree's root, which has no parent. Denote as:

***first-order connections*:** these refer to the connections between nodes in $\mathcal{T}$, and are denoted as $\mathcal{E}_\mathcal{T}$. 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.

[1] Video: https://youtu.be/QDOqNPn9lCk

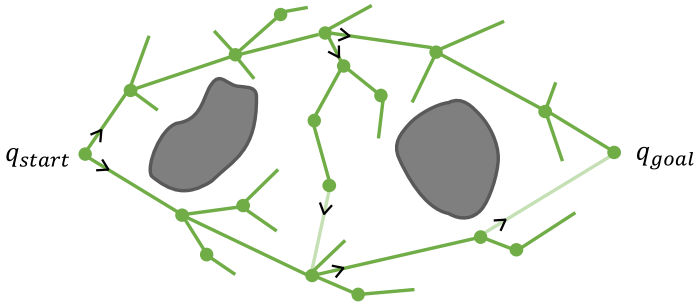


Figure 4.1: MARS leverages two data structures: a tree, represented by green connections, and a graph that expands upon the tree using light green connections. Arrows indicate the direction of travel of the directed graph.

> The set of *second-order connections* is $\text{cl}(\mathcal{E} \setminus \mathcal{E}_{\mathcal{T}})$. $\mathcal{G}$, which extends $\mathcal{T}$, knows about these connections, but $\mathcal{T}$ does not.

This formulation handles multiple paths and interconnecting portions of the same tree. Both $\mathcal{T}$ and $\mathcal{G}$ will be exploited: $\mathcal{T}$ is used to build multiple new solutions, and $\mathcal{G}$ is queried to extract the best one.

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

$$c(\sigma) = \|\sigma\| = \sum_{i=1}^{\mathcal{M}-1} \|q_{i+1} - q_i\| \tag{4.1}$$

where $q_i \in w_\sigma$. 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 (see Chapter 5).

4.2.2 Replanning algorithm at a glance

This section introduces MARS, a path replanner that exploits a set of precomputed paths to find a new solution quickly, even in high-dimensional and complex scenarios. MARS computes a free path when the current

Algorithm 4 MARS: high-level description

1: Define $\mathcal{P}$ as the set of available paths containing those from input set $\mathcal{S}$ and the valid part of σ_{current}
2: Define $\mathcal{Q}_1$ as the queue of the nodes of σ_{current} between q_{current} and the obstacle
3: Sort $\mathcal{Q}_1$ by some criterion
4: **for each** node $q_n \in \mathcal{Q}_1$ **do**
5: Insert the nodes of each $\sigma \in \mathcal{P}$ into a queue $\mathcal{Q}_2$
6: Sort $\mathcal{Q}_2$ by some criterion
7: **for each** $q_j \in \mathcal{Q}_2$ **do**
8: Define the informed set on the best cost up to now
9: Get the subtree rooted in q_n from the informed set
10: Grow the subtree in the informed set to reach q_j
11: Update $\mathcal{Q}_1$ if a solution was found
12: Get the best path from the graph

one becomes infeasible and optimizes it during the execution. It follows an anytime approach so that it gets the first solution quickly and then tries to improve it over time.

Consider Algorithm 4, where a high-level pseudo-code is reported, while the details of the implementations are in Section 4.2.3. The initial step involves incorporating the valid portion of the current path into the pool of available paths given as input and denoted as $\mathcal{S}$. Simultaneously, all nodes along the current path between the robot and the obstacle are placed in a queue, denoted as $\mathcal{Q}_1$. This queue is then sorted based on a defined criterion (*e.g.*, the closest nodes to the robot's current configuration are the first processed). Subsequently, the algorithm endeavors to establish connections between each node $q_n \in \mathcal{Q}_1$ and the nodes from the other available paths, which are organized in a queue labeled $\mathcal{Q}_2$ and sorted according to a specified criterion (*e.g.*, the distance from the node q_n). The next step involves seeking a path connecting a node $q_n \in \mathcal{Q}_1$ to a node $q_j \in \mathcal{Q}_2$. The algorithm begins by establishing an admissible set using the cost of the best solution found thus far. It then identifies the subtree with q_n as its root that lies within this set. Subsequently, it calls upon a sampling-based path planner to expand the subtree until it

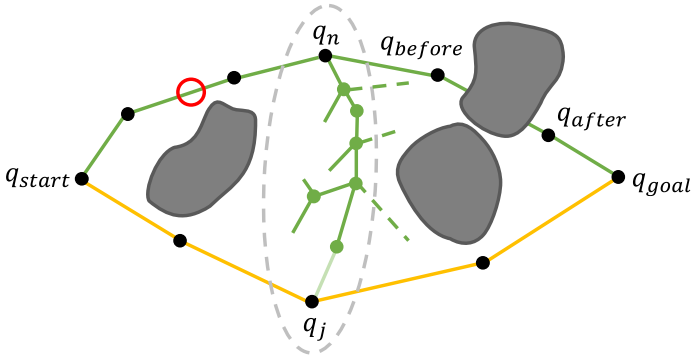
reaches q_j (refer to Figure 4.2a). It is worth noting that this step is not constrained to a specific path planning algorithm. Since q_j possesses a parent node in the tree, a *second-order* connection is employed to link the subtree to q_j. These second-order connections are not visible within the tree but they are in the directed graph. MARS generates multiple solutions from the robot's configuration to the goal, ultimately extracting the optimal one from the graph (as depicted in Figure 4.2b).

4.2.3 Detailed description of the replanning algorithm

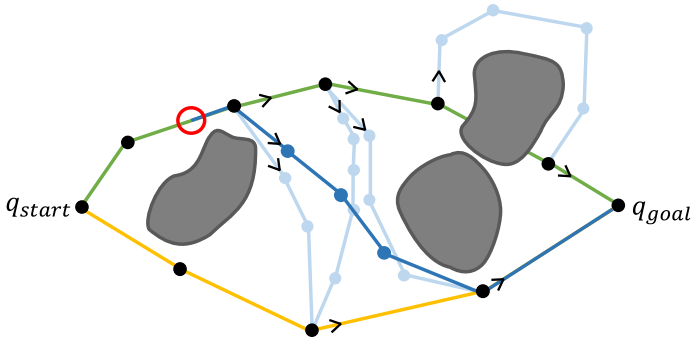
Algorithm 6 is the meta-code of MARS. First, MARS merges the current path tree with those of the available paths (Sub-Algorithm 6a). MARS connects paths that may have been computed with independent trees and builds a graph to extract the best solution. The trees of the available paths need to be merged with that of the current path. Since all trees have the same root (paths start at the same configuration), this procedure is trivial. Ultimately, all paths share the same tree, which belongs to graph $\mathcal{G}$. Figure 4.3 depitcs the procedure.

The solution path $\sigma_{\text{replanned}}$ is initialized with the subpath from q_{current} to the goal. The set of available paths $\mathcal{S}$ given as input is enriched with the valid portion of the subpath $\sigma_{\text{replanned}}$ to form the set $\mathcal{P}$.

All the nodes of $\sigma_{\text{replanned}}[q_{\text{current}}, q_{\text{before}}]$ are inserted into the queue $\mathcal{Q}_1$. Note that q_{before} and q_{after} are provided by the *Collision Check Thread* in Algorithm 3 (Chapter 3). Then, the algorithm computes multiple paths connecting $q_n \in \mathcal{Q}_1$ to the other available paths using `connectToPaths` and assigns the best solution to $\sigma_{q_n q_{\text{goal}}}$; now, the algorithm builds the candidate solution concatenating the subpath from q_{current} to q_n with $\sigma_{q_n q_{\text{goal}}}$. If the candidate solution has a lower cost than $\sigma_{\text{replanned}}$, it becomes $\sigma_{\text{replanned}}$. At this point, $\mathcal{Q}_1$ is updated (*e.g.*, $\mathcal{Q}_1$ is emptied, and the nodes of the new solution not previously used are inserted into the queue). Note that the number of nodes in $\mathcal{Q}_1$ is small, so the sorting function is



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



(b) Simplified representation of MARS outcome. The light blue paths connect the current path to the other available ones. The blue path is the best solution extracted from the graph.

Figure 4.2: Illustration of the subtree rooted in q_n and built to reach q_j. The green path is the current path; the yellow path is another available path. The red circle represents the current robot configuration q_{current}.

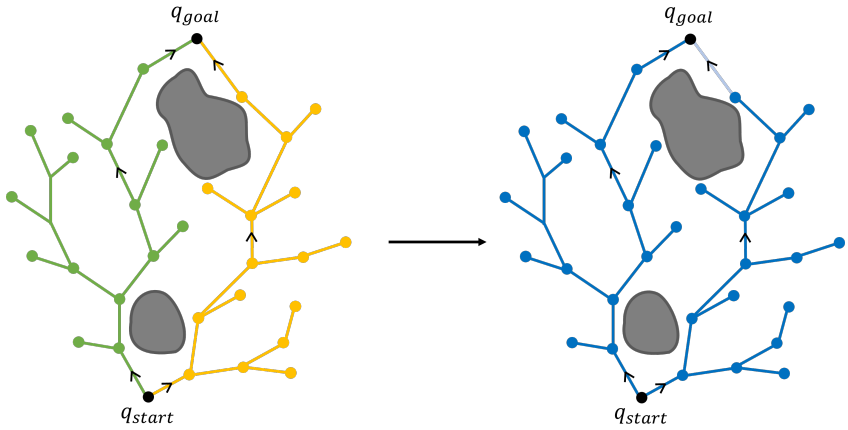
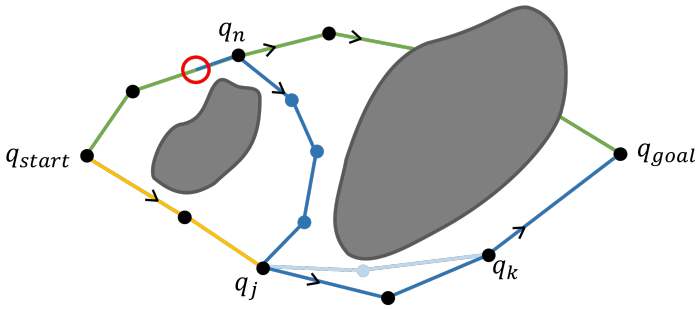


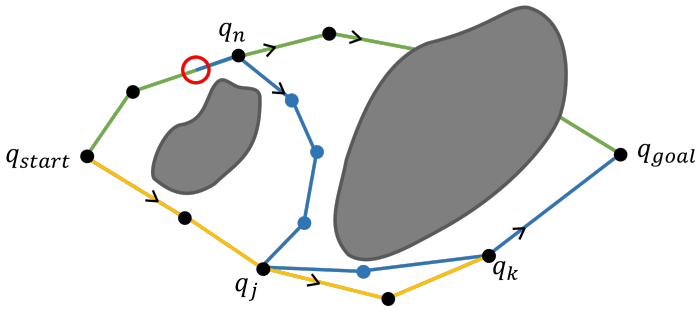
Figure 4.3: Tree merge: The tree associated with the current path (green) combines with that of an alternative path (yellow). Ultimately, a single unified tree emerges, represented in blue. As both paths connect the same goal node, the final connection of the yellow path becomes a *second-order connection* (depicted by the light blue line) within the graph.

not computationally demanding. As shown in Figure 4.4, previous runs of the replanner may have produced segments within the graph $\mathcal{G}$ that enhance the current best solution, such as shortcuts connecting nodes along the same path (indicated by the light blue lines in the figure). Consequently, at the end of the algorithm, if there is any time remaining, MARS initiates a search across the graph $\mathcal{G}$ to identify a path superior to $\sigma_{\text{replanned}}$.

The core of the replanner is `connectToPaths` (Sub-Algorithm 6d), executed for each $q_n \in \mathcal{Q}_1$. This function takes as input $q_n \in \mathcal{Q}_1$ and tries to find paths connecting it to the nodes of the available paths in $\mathcal{P}$. As output, it returns the best path σ_{sol} from q_n to q_{goal} found so far. To do this, first, it populates the queue $\mathcal{Q}_2$ with the nodes of each path $\sigma_j \in \mathcal{P}$ and sorts it based on some criteria (*e.g.*, on the distance from q_n). Then, a path from q_n to each $q_j \in \mathcal{Q}_2$ is computed by `informedPlan`; the path $\sigma_{q_n q_j}$ found is then concatenated with the subpath $\sigma_{q_j q_{\text{goal}}}$ from q_j to q_{goal} if



(a) The best solution built by the algorithm is the blue one, but the graph $\mathcal{G}$ contains a shorter path between q_j and q_k (depicted in light blue) than that taken by the current solution.



(b) The graph search improved the current solution (in blue) by considering the shortest available path between q_j and q_k.

Figure 4.4: MARS generates a solution, but further enhancements are possible by incorporating segments of the graph constructed in earlier iterations. The green, yellow, and blue paths represent the current path, an alternative available path, and the best solution discovered thus far, respectively. The light blue path improves the segment between q_j and q_k compared to the corresponding section of the yellow path and is integrated into the current solution through the graph search.

Algorithm 6 MARS: detailed description

Input: $\sigma_{\text{current}}, q_{\text{current}}, \mathcal{S} = \{\sigma_1, \ldots, \sigma_N\}, \mathcal{G}$, max_time
Ensure: replanned path $\sigma_{\text{replanned}}$
1: mergeTrees$(\sigma_{\text{current}}, \mathcal{S}, \mathcal{G})$
2: $\sigma_{\text{replanned}} \leftarrow \sigma_{\text{current}}[q_{\text{current}}, q_{\text{goal}}]$; $c_{\text{replanned}} \leftarrow c(\sigma_{\text{replanned}})$
3: $\mathcal{P} \leftarrow \mathcal{S}$
4: **if** $c_{\text{replanned}} = \infty$ **then**
5: $\quad \mathcal{P} \xleftarrow{+} \sigma_{\text{replanned}}[q_{\text{after}}, q_{\text{goal}}]$
6: $\quad \mathcal{Q}_1 \leftarrow w_{\sigma_{\text{replanned}}}[q_{\text{current}}, q_{\text{before}}]$
7: **else**
8: $\quad \mathcal{P} \xleftarrow{+} \sigma_{\text{replanned}}$
9: $\quad \mathcal{Q}_1 \leftarrow w_{\sigma_{\text{replanned}}}$
10: $\mathcal{Q}_1$.sortQueue()
11: **while** $\neg$ isEmpty$(\mathcal{Q}_1)$ & $t <$ max_time **do**
12: $\quad q_n \leftarrow \mathcal{Q}_1$.pop()
13: $\quad t_{\text{MAX}} \leftarrow \text{max\_time} - t$
14: $\quad \sigma_{q_n q_{\text{goal}}} \leftarrow$ connectToPaths$(q_n, \sigma_{\text{replanned}}, \mathcal{P}, t_{\text{MAX}})$
15: $\quad$ **if** $\neg$ isEmpty$(\sigma_{q_n q_{\text{goal}}})$ **then**
16: $\quad\quad \sigma_{q_{\text{current}} q_n} \leftarrow \sigma_{\text{replanned}}[q_{\text{current}}, q_n]$
17: $\quad\quad \sigma_{\text{candidate}} \leftarrow \sigma_{q_{\text{current}} q_n} \cup \sigma_{q_n q_{\text{goal}}}$
18: $\quad\quad$ **if** $c(\sigma_{\text{candidate}}) < c_{\text{replanned}}$ **then**
19: $\quad\quad\quad \sigma_{\text{replanned}} \leftarrow \sigma_{\text{candidate}}$; $c_{\text{replanned}} \leftarrow c(\sigma_{\text{candidate}})$
20: $\quad\quad\quad$ solution_found $\leftarrow$ True
21: $\quad\quad\quad \mathcal{Q}_1$.updateQueue$(w_{\sigma_{\text{replanned}}})$
22: **if** solution_found **then**
23: $\quad t_{\text{MAX}} \leftarrow \text{max\_time} - t$
24: $\quad (\sigma_{\text{replanned}}, c_{\text{replanned}}) \leftarrow$ searchBetterPath$(q_{\text{current}}, q_{\text{goal}}, \mathcal{G}, \sigma_{\text{replanned}}, t_{\text{MAX}})$

it results in a better solution than the best one found so far.

The graph search (Sub-Algorithm 6e) 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. Note that MARS takes advantage of the architecture proposed in Algorithm 3, as the *Collision Check*

Sub-Algorithm 6a mergeTrees

Input: σ_{current}, $\mathcal{S} = \{\sigma_1, \ldots, \sigma_N\}$, $\mathcal{G}$
Ensure: updated tree $\mathcal{T}$ and directed graph $\mathcal{G}$
1: $\mathcal{T} \leftarrow \sigma_{\text{current}}$.tree()
2: **for** σ_j in $\mathcal{S}$ **do**
3: $\quad \mathcal{T}_j \leftarrow \sigma_j$.tree()
4: $\quad$ **if** $\mathcal{T}_j \neq \mathcal{T}$ **then**
5: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}_j$
6: **for** σ_j in $\mathcal{S}$ **do**
7: $\quad \sigma_j$.setTree($\mathcal{T}$)
8: **if** $\mathcal{T} \not\subset \mathcal{G}$ **then**
9: $\quad \mathcal{G}$.add($\mathcal{T}$)
10: **return** $\mathcal{T}$, $\mathcal{G}$

Thread checks the entire set of available paths $\mathcal{S}$ and σ_{current}, relieving the replanning algorithm from this task.

The map is computed using a depth-first search algorithm starting from the goal node. Specifically, let be q_{goal} the goal node, q_{start} the start node, q_i the node considered at the *i-th* step of the search, and $c_{q_{\text{goal}}q_i}$ the cost of the branch from q_{goal} to q_i followed during the current iteration. The search along this branch stops when:

$$c_{q_{\text{goal}}q_i} + \|q_i - q_{\text{start}}\| > c_{\text{better}} \tag{4.2}$$

Note that both first-order and second-order connections are considered during the search in the graph $\mathcal{G}$.

The most demanding part of Algorithm 6 is `connectToPaths` (Sub-Algorithm 6d), which must be very efficient. First, we speed up the search by purging the nodes $q_j \in \mathcal{Q}_2$ that can not improve the solution. Let be $q_n \in \mathcal{Q}_1$ and $q_j \in \mathcal{Q}_2$ the the root and goal nodes of the connecting path $\sigma_{q_n q_j}$ (Figure 4.5c). The candidate solution $\sigma_{\text{sol}} = \sigma_{q_n q_j} \cup \sigma_j[q_j, q_{\text{goal}}]$ is

Sub-Algorithm 6b growInInformedSet

Input: $\mathcal{T}$, $\mathcal{I}$, q_j, t_{MAX}
Ensure: solution_found
1: **while** $\neg$ solution_found & iter $<$ max_iter & $t < t_{\text{MAX}}$ **do**
2: iter $\leftarrow$ iter $+ 1$
3: $q \leftarrow$ sampleInformedSet($\mathcal{I}$)
4: $q_{\text{new}} \leftarrow \mathcal{T}$.growTree($q$)
5: **if** $||q_{\text{new}} - q_j|| <$ min_dist **then**
6: **if** path from tree's root to q_j is valid **then**
7: create a second-order connection from q_{new} to q_j
8: solution_found $\leftarrow$ True
9: **else**
10: hide the invalid branch temporarily from $\mathcal{T}$
11: **return** solution_found

better than the current best solution $\sigma_i[q_n, q_{\text{goal}}]$, if:

$$c(\sigma_{q_n q_j}) < c(\sigma_i[q_n, q_{\text{goal}}]) - c(\sigma_j[q_j, q_{\text{goal}}]) \tag{4.3}$$

The lower bound of $c(\sigma_{q_n q_j})$ is the Euclidean distance from q_n to q_j. So, q_j can lead to an improved solution if:

$$\left\|q_n - q_j\right\| < c(\sigma_i[q_n, q_{\text{goal}}]) - c(\sigma_j[q_j, q_{\text{goal}}]) \tag{4.4}$$

If Equation (4.4) does not hold, q_j is discarded by `connectToPaths` (line 9). Note that, if $\sigma_i[q_n, q_{\text{goal}}]$ is infeasible, Equations (4.3) and (4.4) always hold as the path has infinite cost. When a solution is found, Equation (4.4) is updated with the new path's cost $c(\sigma_{\text{sol}})$, such that:

$$\left\|q_n - q_j\right\| < c(\sigma_{\text{sol}}) - c(\sigma_j[q_j, q_{\text{goal}}]) \tag{4.5}$$

In this way, only nodes in $\mathcal{Q}_2$ with a greater than zero probability of improving the current solution are considered.

To enhance the speed of Sub-Algorithm 6d, we also exploit *informed*

Sub-Algorithm 6c informedPlan

Input: q_n, q_j, $c_{\max}$, max_time
Ensure: $\sigma_{q_n q_j}$ from q_n to q_j and its cost
1: $\mathcal{I} \leftarrow$ informed set defined by $c_{\max}$
2: $\mathcal{T}_s \leftarrow$ subtree rooted in q_n contained in $\mathcal{I}$
3: $\mathcal{G}_{\mathcal{T}_s} \leftarrow$ the subgraph of $\mathcal{G}$ containing $\mathcal{T}_s$
4: $(\sigma_{q_n q_j}, c_{q_n q_j}) \leftarrow$ searchBetterPath$(q_n, q_j, \mathcal{G}_{\mathcal{T}_s}, c_{\max}, \text{max_time})$
5: **if** $\neg$ isEmpty$(\sigma_{q_n q_j})$ **then**
6: solution_found $\leftarrow$ True
7: **else**
8: $t_{\text{MAX}} \leftarrow$ max_time $- t$
9: solution_found $\leftarrow$ growInInformedSet$(\mathcal{T}_s, \mathcal{I}, q_j, t_{\text{MAX}})$
10: **if** solution_found **then**
11: $(\sigma_{q_n q_j}, c_{q_n q_j}) \leftarrow$ searchBetterPath$(q_n, q_j, \mathcal{G}_{\mathcal{T}_s}, c_{\max})$
12: **return** $\sigma_{q_n q_j}$, $c_{q_n q_j}$, solution_found

sampling [47] in `informedPlan` (Sub-Algorithm 6c). When searching for a path connecting q_n to q_j, the search space can be shrunk to the following prolate hyperspheroid:

$$\mathcal{I} = \{q \in \mathcal{C}_{\text{free}} \mid \|q - q_n\| + \|q_j - q\| < c_i\} \tag{4.6}$$

where $c_i = c(\sigma_{\text{sol}}) - c(\sigma_j[q_j, q_{\text{goal}}])$. Equation (4.6) represents an admissible set, so the nodes outside the hyperspheroid are discarded since they cannot improve the solution (Figure 4.5). `informedPlan` checks if an existing path between q_n and q_j with a cost lower than $c_{\max}$ exists. This search is in $\mathcal{G}_{\mathcal{T}_s}$, *i.e.*, the subgraph that contains the *first-order* and the *second-order connections* between nodes of $\mathcal{T}_s$ or to q_j. A ready-to-use solution can exist thanks to previous replanning iterations. Otherwise, `growInInformedSet` invokes `growTree` that grows the subtree in the hyperspheroid $\mathcal{I}$ to reach q_j. `growTree` is a generic sampling-based planner, and the approach is easily extendable to bi-directional ones. If a solution is found, replacing the connection to q_j with a *second-order connection* is the only caveat.

Sub-Algorithm 6d conntectToPaths

Input: $q_n, \sigma_i, \mathcal{P}$, max_time
Ensure: a new path σ_{sol} from q_n to q_{goal}
$\sigma_{\text{sol}} \leftarrow \sigma_i[q_n, q_{\text{goal}}]$; $c_{\text{sol}} \leftarrow c(\sigma_{\text{sol}})$
for $\sigma_j \in \mathcal{P}$, $q_j \in w_{\sigma_j}$ **do**
  $\mathcal{Q}_2$.addTuple(q_j, σ_j)
$\mathcal{Q}_2$.sortQueue()
while $\neg$ isEmpty$(\mathcal{Q}_2)$ & $t <$ max_time **do**
  $(q_j, \sigma_j) \leftarrow \mathcal{Q}_2$.pop()
  $\sigma_{q_j q_{\text{goal}}} \leftarrow \sigma_j[q_j, q_{\text{goal}}]$; $c_{q_j q_{\text{goal}}} \leftarrow c(\sigma_{q_j q_{\text{goal}}})$
  $c_{\max} \leftarrow c_{\text{sol}} - c_{q_j q_{\text{goal}}}$
  if $\left\| q_n - q_j \right\| < c_{\max}$ **then**
    $t_{\text{MAX}} \leftarrow$ max_time $- t$
    $\sigma_{q_n q_j}, c_{q_n q_j}$, solution_found $\leftarrow$ informedPlan$(q_n, q_j, c_{\max}, t_{\text{MAX}})$
    if solution_found **then**
      $c_{\text{new}} \leftarrow c_{q_n q_j} + c_{q_j q_{\text{goal}}}$
      if $c_{\text{new}} < c_{\text{sol}}$ **then**
        $\sigma_{\text{sol}} \leftarrow \sigma_{q_n q_j} \cup \sigma_{q_j q_{\text{goal}}}$; $c_{\text{sol}} \leftarrow c_{\text{new}}$
return σ_{sol}

`growInInformedSet` adopts a lazy collision check approach. The connections already in the subtree are checked only once a path is found. The branches with a collision are hidden, and growth begins again.

Figure 4.5 reports one iteration of `connectToPaths`. We select $q_j \in \mathcal{Q}_2$ as a valid node to try to connect to by exploiting $\sigma_i[q_n, q_{\text{goal}}]$ and $\sigma_j[q_j, q_{\text{goal}}]$ (the bold green and yellow lines). The hyperspheroid $\mathcal{I}$ (grey dotted ellipse) is defined by Equation (4.6), and the already existing subtree rooted in q_n and contained in $\mathcal{I}$ is selected (Figure 4.5a). The connections of the subtree outside $\mathcal{I}$ are not considered (dotted green lines). Then, the subtree is grown in $\mathcal{I}$ using a path planner (Figure 4.5b). Once q_j is reached, a second-order connection is created between it and the subtree (light green line). The light blue path (Figure 4.5c) is the path $\sigma_{q_n q_j}$ found. If the cost of $\sigma_{q_n q_j} \cup \sigma_j[q_j, q_{\text{goal}}]$ is less than the cost of $\sigma_i[q_n, q_{\text{goal}}]$, it becomes the new candidate solution. Then, the procedure is repeated for each $q_j \in \mathcal{Q}_2$ that satisfies Equation (4.5). Once the whole $\mathcal{Q}_2$ has been

Sub-Algorithm 6e searchBetterPath

Input: $q_s, q_g, \mathcal{P}, \sigma_c$ OR $c_{\max}, t_{\text{MAX}}$
Ensure: σ_{better} from q_s to q_g and its cost
1: **if** $\neg$ isEmpty(σ_c) **then**
2: $\quad \sigma_{\text{better}} \leftarrow \sigma_c$; $c_{\text{better}} \leftarrow c(\sigma_c)$
3: **else**
4: $\quad \sigma_{\text{better}} \leftarrow$ VOID; $c_{\text{better}} \leftarrow c_{\max}$
5: **if** t_{MAX} is VOID **then**
6: $\quad t_{\text{MAX}} \leftarrow \infty$
7: sorted_map $\leftarrow \mathcal{G}$.lowerCostPaths$(q_s, q_g, c_{\text{better}}, t_{\text{MAX}})$
8: **while** $\neg$ solution_found & $\neg$ isEmpty(sorted_map) **do**
9: $\quad (\sigma_{\text{new}}, c_{\text{new}}) \leftarrow$ sorted_map.pop()
10: $\quad$ **if** σ_{new}.valid() **then**
11: $\quad\quad \sigma_{\text{better}} \leftarrow \sigma_{\text{new}}$; $c_{\text{better}} \leftarrow c_{\text{new}}$
12: $\quad\quad$ solution_found $\leftarrow$ True
13: **return** $\sigma_{\text{better}}, c_{\text{better}}$

considered, $\mathcal{Q}_1$ is updated, and a new q_n popped.

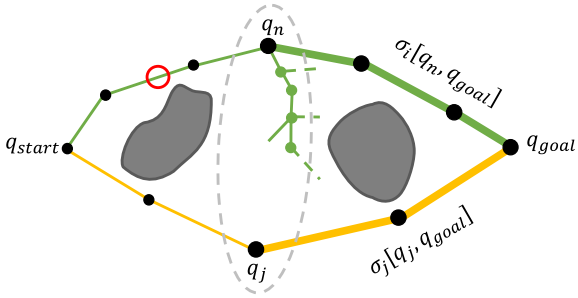
4.2.3.1 Considerations on the available paths

MARS relies on pre-calculated paths to find and optimize a solution over time, and they play a crucial role in shaping the algorithm's performance.

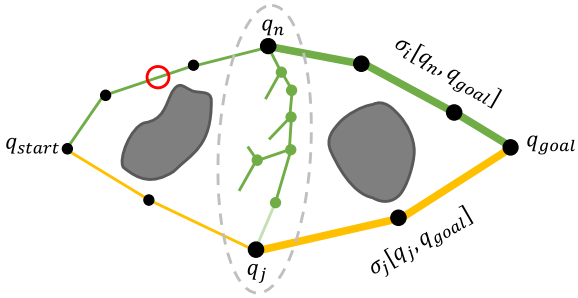
In principle, the higher the number of available paths, the better chance MARS has of finding and optimizing a solution. However, a surplus of paths results in more iterations of the `connectToPaths` algorithm, driven by the expanded set of nodes $\mathcal{Q}_2$.

Moreover, paths that are excessively optimized may exhibit considerable similarity. Consequently, a single obstacle could render significant portions of all available paths impractical, impeding the rapid discovery of a feasible solution. Conversely, paths that are inadequately optimized may lead MARS to uncover solutions of inferior quality.

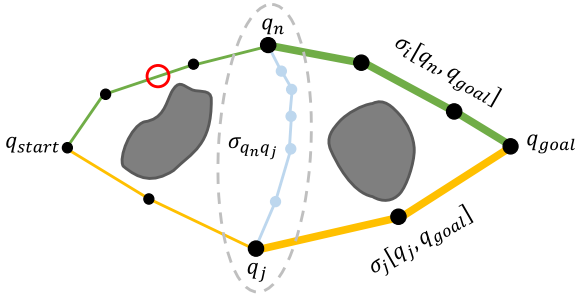
Both the quantity and the quality of available paths impact the algorithm's performance. Striking a balance is crucial to ensure an adequate



(a) The subtree rooted in q_n belonging to the ellipse $\mathcal{I}$ of Equation (4.6). The dotted green line are not considered because not in $\mathcal{I}$.



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



(c) The connecting path from q_n to q_j found (light blue path).

Figure 4.5: Subtree (green) and subpaths (bold lines) considered to reduce the computational load of connectToPaths.

number of paths without overburdening computational resources and to identify paths covering diverse segments of the search space. Although these aspects deserve further investigation, this thesis limits on experimentally delineating these trade-offs.

Typically, having $2 - 4$ additional paths alongside the robot's current one is sufficient to achieve satisfactory performance.

Moreover, although not infallible, it has been observed that applying some iterations of local optimizations, such as localized rewiring or shortcutting methods, to paths generated by non-optimal algorithms like RRT or RRT Connect is generally effective in computing a diverse set of paths. It is essential to underscore that these considerations pertain solely to the calculation of the set of alternative paths $\mathcal{S}$ and not to the initial path for the robot, which is generally intended to be at least near-optimal.

4.2.4 Properties of the replanning algorithm

Sampling-based path planners are considered probabilistically complete when the likelihood of discovering a solution is equal to one with an infinite number of samples. Furthermore, if the cost of the solution converges to the optimal value as the sample count tends towards infinity, these planners are deemed almost-surely asymptotically optimal. For formal definitions, please refer to Section 2.3.4.1.

It is important to note that these properties cannot be assured for replanners without any assumptions regarding obstacle dynamics. Indeed, it is possible to construct scenarios where an obstacle consistently blocks any newly provided solution, rendering it impossible for the robot to find a path to the goal. However, by assuming a static scene starting from a specific point in time onwards, we can establish and demonstrate these properties for MARS.

4.2.4.1 Probabilistic completeness

MARS searches for a path from a generic start node $q_n \in \mathcal{Q}_1$ to a generic goal node $q_j \in \mathcal{Q}_2$, which do not necessarily correspond to the current robot configuration and the goal. Note that the subpaths from the robot configuration to q_n and from q_j to the goal are always feasible (otherwise Equation (4.5) in Section 4.2.3 does not hold). The problem, therefore, reduces to proving that the algorithm is complete in searching for a path from q_n to q_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 Equation (4.6) in Section 4.2.3). The hyperspheroid, 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.

4.2.4.2 Asymptotic optimality

MARS achieves asymptotic optimality by employing an asymptotically optimal planner, such as RRT*, to grow the subtree in Sub-Algorithm 6b. When MARS tries to optimize the current path, it chooses pairs of nodes (q_n, q_j) such that $q_n \in \mathcal{Q}_1$ and $q_j \in \mathcal{Q}_2$. When q_n corresponds to the current robot configuration and q_j corresponds to the goal, the algorithm behaves like Informed RRT* [49] and is therefore asymptotically optimal. For all other pairs of nodes, MARS optimizes the connecting path between the two.

4.3 Experimental results

4.3.1 Challenges in benchmarking

Experimental tests are crucial for evaluating algorithms, often involving comparisons with established state-of-the-art methods. However, the results are greatly influenced by the simulated environment in which the robot operates and the chosen evaluation metrics. This section succinctly outlines the complexities and critical considerations involved in conducting a benchmarking campaign for replanning algorithms.

4.3.1.1 Test design

The definition of the scenario is a critical step in testing, with algorithm performance susceptible to fluctuations based on factors such as the complexity of the search space and the variability of the environment:

- Numerous elements contribute to the complexity of the problem. These include the size and dimensionality of the search space, the proportion of space occupied by fixed obstacles, and the presence of narrow passages. These aspects can significantly impact the performance of a replanning algorithm. An algorithm that excels in a small and obstacle-sparse environment may see a sharp decline in performance when faced with a larger space. Escalating the dimensionality of the search space amplifies the complexity exponentially. Graph-based algorithms, for instance, are notably susceptible to the curse of dimensionality, a challenge that also extends to sampling-based replanners.

- An adequate assessment of a replanning algorithm should consider the variability in both the problem settings and the dynamic nature of the environment. Evaluating the algorithm based on just one

set of start and goal configurations while assuming consistent obstacle behaviour offers an incomplete measure of its performance. Instead, multiple tests with several sets of start and goal configurations and a diverse range of moving obstacles in terms of quantity, trajectories, and sizes should be considered. Quantity of obstacles significantly impacts the algorithm's efficiency. For example, DRRT [40] demonstrates the capability to adapt its tree structure dynamically in response to obstacles. However, when faced with many obstacles, a substantial portion of the tree may be invalidated, making it challenging to find a new solution within a reasonable planning timeframe.

Generally, a path replanning algorithm must be capable of rectifying the current path within a limited timeframe, ranging from tens to a few hundred milliseconds. The maximum allowed execution time for the algorithm significantly influences the outcomes. An excessively small value may not afford sufficient time to find a solution, while an overly large value could render the robot unresponsive. Determining the maximum time requires careful calibration, striking a balance between the robot's responsiveness and the algorithm's ability to find a solution. The complexity of the specific problem at hand significantly influences this calibration process.

4.3.1.2 Metrics design

Benchmarking results are contingent on the metrics employed to evaluate replanning algorithms. It is imperative to establish comprehensive and representative metrics that facilitate fair and meaningful comparisons.

Standard motion planning metrics encompass planning success rate (*i.e.*, whether the planner successfully finds a solution), number of iterations (or planning time) required to find a solution, and path cost (typically represented by the path length). Given that the primary objective of

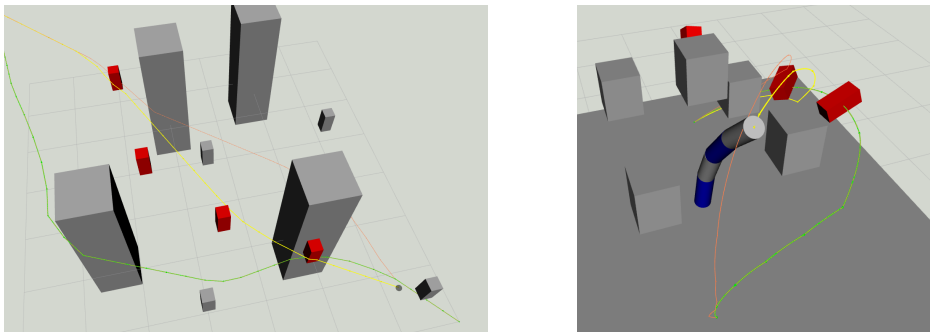
a replanning algorithm is to ensure the robot reaches its destination while avoiding collisions, the *success rate (S%)* emerges as a paramount metric:

$$S\% = \frac{\text{tests without collisions}}{\text{total number of tests}} \tag{4.7}$$

Furthermore, it is crucial to assess algorithms based on the quality of the solutions they produce. When success rates are comparable or reach similar values, the algorithm should be selected to yield a higher path quality. A standard metric for evaluating path quality is its length. However, unlike path planning, with path replanning the path dynamically adjusts to avoid moving obstacles. Consequently, a relevant performance measure becomes the length of the path actually traversed.

For a fair comparison among different tests, this length should be normalized with respect to the optimal path length. However, calculating or even defining what the optimal path is in a context where obstacles may appear and disappear multiple times along the robot's trajectory can be complex. Therefore, an alternative approach could be normalizing with respect to the length of the initial path.

Occasionally, the time an algorithm takes to furnish an initial viable solution is considered a valuable metric. Certain algorithms, like DRRT [40], cease execution once a feasible solution is found. Conversely, others, like Anytime DRRT [42], utilize all available time not only to find a feasible solution but also to enhance it. Nonetheless, it is crucial that a replanning algorithm delivers a solution within a user-defined maximum time frame, coupled with a high-quality solution. Merely providing a solution in 50 ms, with a maximum time limit set at 200 ms, is less relevant if the algorithm does not effectively utilize the remaining time.



(a) Medium 3-DoF scenario. **(b)** 6-DoF scenario.

Figure 4.6: 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 moving ones.

Table 4.1: Settings of the different scenarios used for MARS benchmarking.

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

4.3.2 Numerical simulations

Simulations were conducted to compare MARS with some state-of-the-art path replanners. In particular, MARS was compared with DRRT [40], Anytime DRRT [42], MPRRT [157], and with [27], which we will refer to by the acronym DRRT* for convenience. The tests were conducted on a computer with a 2.80 GHz CPU using ROS [138] and *MoveIt!* [69] for scene management and collision checking. The replanners were implemented in C++ using OpenMORE [161] (refer to Chapter 6) and integrated into the architecture of Algorithm 3. The code can be found at [160]. Tests were carried out in accordance with the principles outlined in Section 4.3.1. The algorithms were tested in 6 scenarios, which differ in the size and dimensionality of the search space (DoF 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 4.1);

Scenarios 4, 5 and 6 provide a 6-, 12-, and 18-DoF robotic manipulator, 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's 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 [87], followed by an optimization procedure [80]. During the execution, new obstacles randomly appear, obstructing the robot's path and forcing the replanner to find a new solution. Table 4.1 shows the number of fixed and moving obstacles. Each algorithm is assigned a maximum replan time of 200 ms. The frequency of the *Trajectory Execution Thread* and the *Collision Checking Thread* are 500 Hz, and 30 Hz. The queue $\mathcal{Q}_1$ of MARS is sorted in descending

order based on the distance along the path from the current robot configuration. The queue $\mathcal{Q}_2$ is sorted in ascending order based on the distance from the processed node $q_n \in \mathcal{Q}_1$.

The algorithms are evaluated using the following metrics:

Success rate ($S\%$): denote N_{success} and N_{tests} as the number of tests executed without collisions and the number of tests performed in each scenario, respectively. The success rate:

$$S\% = \frac{N_{\text{success}}}{N_{\text{tests}}} \tag{4.8}$$

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

Collision rate ($C\%$): 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:

$$C\% = \frac{N_{\text{collisions}}}{((1 - S\%)\, N_{\text{tests}}\, N_{obs})} \tag{4.9}$$

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 ($n.p.l.$): let $\|\sigma_{\text{real}}\|$ and $\|\sigma_{\text{init}}\|$ denote the actual path length traversed by the robot and the initially computed path length at the beginning of the iteration, respectively. The equation:

$$n.p.l. = \frac{\|\sigma_{\text{real}}\|}{\|\sigma_{\text{init}}\|} \tag{4.10}$$

Table 4.2: Success rate ($S\%$) and Collision rate ($C\%$) of the replanning algorithms in the different scenarios tested. Maximum replanning time 200 ms.

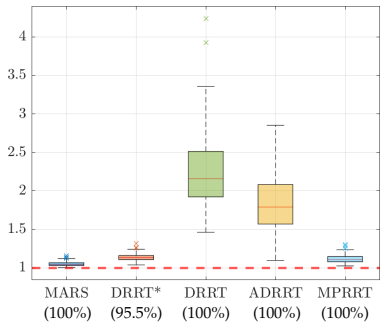
	Small 3-DoF		Medium 3-DoF		Large 3-DoF	
	$S\%$	$C\%$	$S\%$	$C\%$	$S\%$	$C\%$
MARS	**100.0**	**0.0**	**100.0**	**0.0**	**100.0**	**0.0**
DRRT*	95.5	33.3	89.5	19.0	52.0	34.8
DRRT	100.0	0.0	18.0	55.2	1.0	79.3
Anytime DRRT	100.0	0.0	17.5	56.3	1.50	80.2
MPRRT	100.0	0.0	79.5	23.2	11.0	29.3

	6-DoF		12-DoF		18-DoF	
	$S\%$	$C\%$	$S\%$	$C\%$	$S\%$	$C\%$
MARS	**94.0**	**38.9**	**93.0**	**36.6**	**88.5**	**37.3**
DRRT*	66.0	44.8	9.0	77.1	4.0	82.2
DRRT	0.0	98.9	0.0	99.7	0.0	99.8
Anytime DRRT	0.0	99.4	0.0	99.3	0.0	99.7
MPRRT	2.0	92.7	7.5	76.4	9.0	72.2

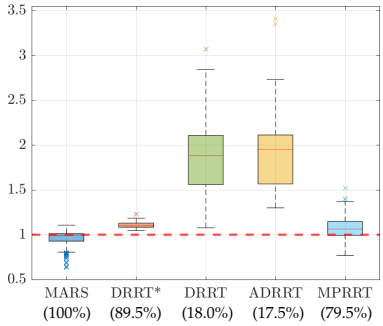
indicates how longer (due to obstacle avoidance) or shorter (due to path optimization) the path is compared to the initial one. Since σ_{real} depends on σ_{init}, its length is normalized to obtain a comparable measure across tests.

Figure 4.6 shows screenshots of two tests of MARS in a 3-DoF and 6-DoF scenario. Table 4.2 shows the success rate $S\%$ and collision rate $C\%$ of the algorithms in each scenario. Figure 4.7 shows the boxplot of the normalized path length $n.p.l.$ for replanners that achieved at least a 5% success rate.

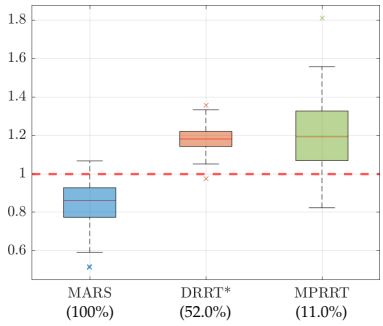
All planners have a high success rate for scenario 1. As the complexity grows, MARS maintains a high success rate ($S\% \geq 88.5\%$). In scenarios 5 and 6, MPRRT and DRRT* achieve a success rate of 9.0% and 4.0%, respectively, whereas DRRT and Anytime DRRT always fail. MPRRT performs better than DRRT and Anytime DRRT in scenario 6 because



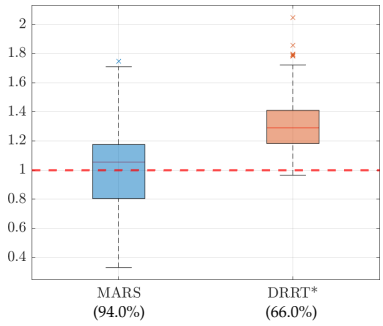
(a) Small 3-DoF scenario.



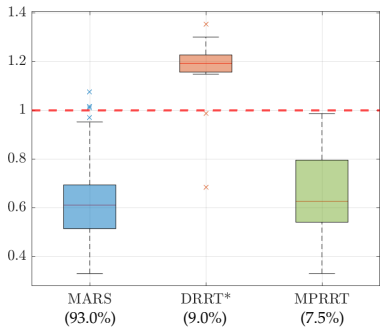
(b) Medium 3-DoF scenario.



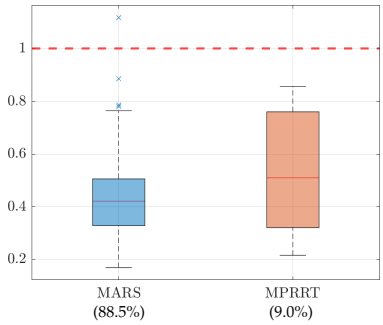
(c) Large 3-DoF scenario.



(d) 6-DoF scenario.



(e) 12-DoF scenario.



(f) 18-DoF scenario.

Figure 4.7: Normalized path length (*n.p.l.*) in each scenario. A boxplot below the red dashed line indicates the replanner generated shorter paths than the initial one. The success rate is listed under each replanner's name. Only replanners with a success rate exceeding 5% are shown. Maximum replanning time 200 ms.

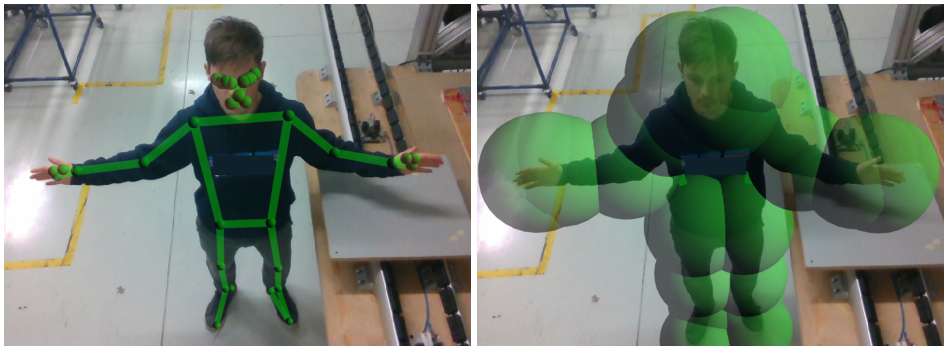
Table 4.3: Overview of the properties of the tested algorithms.

Algorithm	Replan	Anytime	Asymptotic optimality
MARS	Always	✓	When coupled with an optimal planner
DRRT	If path is obstructed	✗	✗
Anytime DRRT	Always	✓	✗
MPRRT	Always	✓	As computation power increases
DRRT*	If path is obstructed	✓	✗

it plans on multiple parallel threads, and its collision rate remains lower than DRRT and Anytime DRRT.

Table 4.3 offers an overview of algorithms properties influencing the quality of the solutions they provide. Notably, while DRRT and DRRT* are selectively triggered only when the path is obstructed, Anytime DRRT, MPRRT and MARS persistently search for improved solutions even when there are no obstacles on the current path. Each algorithm adopts an anytime approach, quickly generating an initial solution and allocating any remaining time within the same replanning cycle for further refinement. The sole exception is DRRT, which terminates upon discovery of a feasible solution. Moreover, certain algorithms exhibit the capacity to furnish optimal solutions within environments that can be regarded as static from a particular point in time onward. Section 4.2.4.2 shows that MARS is asymptotically optimal when coupled with an asymptotically optimal planner. Additionally, [157] proves that MPRRT tends towards the optimal solution as parallel computing power increases.

Figure 4.7 shows the $n.p.l.$ for the algorithms with a $S\% \geq 5\%$. When a boxplot is below the red dashed line, it means that the replanner produced shorter paths than the one computed initially. In scenarios 1-2

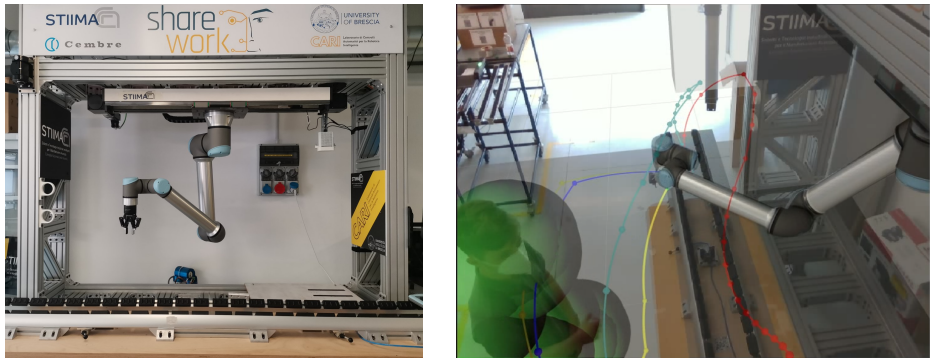


(a) Human skeleton key-points.

(b) Each key-point is associated with a sphere, considered as a dynamic obstacle.

Figure 4.8: The vision system, coupled with a human skeleton tracking algorithm, identifies the presence of a human within the robotic cell and delineates specific key-points on the skeleton. These key-points, represented as Cartesian coordinates, serve a dual purpose: they allows SSM computation and act as the center for spheres considered as dynamic obstacles by the replanner.

MARS, DDRT* and MPRRT provide comparable *n.p.l.* (Figure 4.7a and 4.7b). 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 short. 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).



(a) Robotic cell. **(b)** Test on the real cell.

Figure 4.9: Collaborative robotic cell featuring a UR10e mounted upside down and Intel RealSense D435.

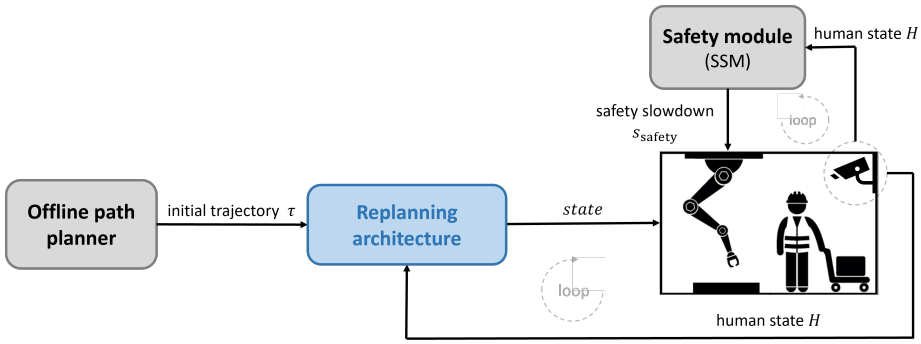


Figure 4.10: Planning scheme overview: The offline planner computes the initial trajectory and a set of paths for MARS. The path replanning module executes the robot's trajectory while seeking enhanced solutions. The speed modulation module adjusts the robot's velocity in compliance with SSM from ISO/TS 15066.

4.3.3 Real-world experiments

MARS and the replanning architecture were tested within the collaborative robotic cell illustrated in Figure 4.9. The setup features a 6-DoF UR10e manipulator mounted upside down and an Intel Realsense D435 for tracking the operator's position. The experiments involve the robot moving from an initial position to a target position. Simultaneously, an operator obstructs the initially planned path, prompting the algorithm to search for an alternate valid path and optimize it over time.

4.3.3.1 Software architecture

The planning process, outlined in Figure 4.10, involves a motion planner to compute the initial trajectory and a set of paths for MARS. The replanning architecture (detailed in Chapter 3) orchestrates the robot's motion while concurrently executing the replanning algorithm. This system receives real-time updates regarding the operator's pose via the vision system, which utilizes a human skeleton tracking algorithm [145] based on MediaPipe BlazePose [5]. This algorithm provides Cartesian coordinates corresponding to key-points on the human skeleton (Figure 4.8a). Each key-point serves as the center for a sphere encapsulating a segment of the human body, with the sphere's radius customizable by the user. Together, these spheres represent the dynamic obstacles within the operational environment (Figure 4.8b).

Additionally, a safety module based on SSM from ISO/TS 15066 regulates the robot's velocity in response to its proximity to the human.

The entire framework has been realized within the ROS ecosystem [138], where the human tracking system, the SSM module, and the replanning architecture operate as distinct ROS nodes. *MoveIt!* [69] maintains the planning scene updated with the dynamic spherical obstacles derived from the vision system's output (Figure 4.8b) and provides colli-

sion checking capabilities for the replanning algorithm. Specifically, the replanning architecture is actually implemented by OpenMORE [161] (refer to Chapter 6). This architecture continually generates the subsequent robot command, which is transmitted to the robot's controller utilizing the ROS Control [24] framework.

4.3.3.2 SSM safety module

The SSM module dynamically adjusts the robot's velocity as per Equation (2.21) to ensure safety. It implements a slight variation of the dynamic SSM 2D (dSSM) introduced in [12]. The module considers a set of m keypoints on the human skeleton $\mathcal{H} = \{h_1, h_2, \ldots, h_m\}$ and a set of p keypoints on the robot structure $\mathcal{R} = \{r_1, r_2, \ldots, r_p\}$, where $h_k, r_j \in \mathbb{R}^3$ are Cartesian coordinates relative to a common reference frame. Each point $r_j \in \mathcal{R}$ is associated with the robot's configuration q through the forward kinematic function $r_j = \text{fk}_j(q)$, while the points $h_k \in \mathcal{H}$ are provided by the vision system and the human skeleton tracking algorithm (Figure 4.8a). The minimum human-robot distance, denoted as S in Equation (2.21), is computed as:

$$S = \min_{\substack{k=1,\ldots,m \\ j=1,\ldots,p}} \|r_j - h_k\| \tag{4.11}$$

Algorithm 7 easily solves Equation (4.11).

Let u_{r_j,h_k} represent the unit vector from r_j to h_k, with $\dot{h}_k$ and $\dot{r}_j$ denoting the velocities of h_k and r_j respectively. The velocity v_{r_j,h_k} of the robot point r_j toward the human point h_k is determined by:

$$v_{r_j,h_k} = (\dot{r}_j - \dot{h}_k) \cdot u_{r_j,h_k} = (J_j(q)\dot{q} - \dot{h}_k) \cdot u_{r_j,h_k} \tag{4.12}$$

where $J_j(q)$ signifies the Jacobian of r_j. The maximum velocity of the

Algorithm 7 Minimum human-robot distance computation

Input: $\mathcal{H} = \{h_1, h_2, \ldots, h_m\}$, $\mathcal{R} = \{r_1, r_2, \ldots, r_p\}$
Ensure: minimum human-robot distance S

```
S ← ∞
for j = 1 to p do
    for k = 1 to m do
        S' ← ‖r_j − h_k‖
        if S' < S then
            S ← S'
return S
```

robot towards the human is:

$$v_{rh} = \max_{\substack{k=1,\ldots,m \\ j=1,\ldots,p}} (v_{r_j,h_k}) \tag{4.13}$$

At each instant, the safety speed slowdown $s_{\text{safety}} \in [0, 1]$ is employed to scale the robot's velocity and it is calculated as:

$$s_{\text{safety}} = \begin{cases} \min\left(\frac{v_{\max}}{v_{rh}}, 1\right) & \text{if } v_{rh} > 0 \\ 1 & \text{if } v_{rh} \leq 0 \end{cases} \tag{4.14}$$

where $v_{\max}$ is computed using Equation (2.21). When $v_{rh} \leq 0$ the robot is moving away from the human, and thus no velocity scaling is required.

4.3.3.3 Discussion

The replanning algorithm adeptly adjusts the robot's path in response to obstructions caused by the operator. However, its inclination towards the shortest path leads to frequent decelerations imposed by the safety module, as the generated solutions often bring the robot into close proximity to the operator. Consequently, while the robot consistently follows a collision-free path, it may not always select the most efficient one for

collaborative scenarios. This emphasizes the crucial importance of integrating human awareness into the replanning process, considering how the safety module influences generated solutions within the context of human-robot collaboration. This issue will be thoroughly addressed in Chapter 5.

4.4 Summary

This chapter introduced MARS, a novel sampling-based path replanning algorithm for robots in dynamic, high-dimensional search spaces. What sets MARS apart is its utilization of pre-computed paths to swiftly generate and optimize a new path when the current one is obstructed by unforeseen obstacles. MARS simplifies the search process by connecting the current path to nodes from available paths that are closer to the current path than the final goal. By assessing the cost of the current solution, the algorithm selects nodes from existing paths to connect with, progressively constructing a directed graph to leverage efforts from prior iterations. Through the incorporation of informed sampling, subtree reuse, and efficient collision checks, MARS significantly reduces computational overhead, enabling faster and more refined solutions.

Comparative evaluations were conducted pitting MARS against four prominent sampling-based replanning algorithms across diverse high dimensional scenarios. Simulations involved moving a robot within an environment populated with dynamic obstacles capable of suddenly invalidating the robot's intended path. The findings affirm MARS superiority over the baseline algorithms, demonstrating higher success rates and superior solution quality, particularly in scenarios involving high-DoF robots.

In addition to simulation-based testing, we also evaluated the algorithm's performance in a real-world collaborative setting, employing a

6-DoF robot. The algorithm demonstrated good proficiency in dynamically replanning the robot's path. Nonetheless, the generated solutions necessitated frequent safety slowdowns, owing to their proximity to the human operator. This issue motivated the work of Chapter 5.

CHAPTER 5

Human-aware motion replanning

This chapter explores the challenge of motion replanning in human-robot collaborative scenarios, placing emphasis on both reactivity and safety-compliant efficiency.

In collaborative environments, it is imperative for motion planners to factor in potential slowdowns introduced by safety protocols. To enhance collaboration efficiency between humans and robots, in the first part of the chapter we combine a responsive path replanning algorithm with a safety-aware cost function. This empowers the robot to dynamically adjust its path in real-time, reacting to shifts in the human's state considering the safety-related effects. The resultant algorithm, called **H**uman-**A**ware **M**ulti-p**A**th **R**eplanning **S**trategy (MARSHA), minimizes the necessity for trajectory slowdowns, subsequently reducing execution time and enhancing overall operational efficiency.

In the second part of the chapter we conduct a comprehensive series of simulations and real-world experiments in an industrial collaborative robotic environment to validate our approach. The outcomes exhibit the superior efficacy of our method in comparison to conventional

human-robot cooperation techniques, showcasing notable enhancements in terms of both efficiency and safety.
This chapter is based on the work presented in the submitted paper [162].

5.1 Introduction

Collaborative setups introduce inherent risks for human operators. It is imperative to conduct a comprehensive risk analysis to pinpoint and mitigate potential hazards. Factors such as the frequency and energy of human-robot interactions are pivotal in this assessment [170]. As discussed in Section 2.4, the ISO/TS 15066 safety standard (*Robots and robotic devices – Collaborative robots* [70]) guides roboticists in crafting collaborative operations, including the *Power and Force Limiting* (PFL) and the *Speed and Separation Monitoring* (SSM) modes, which delineate safety requirements for applications with and without physical contact, respectively.

In typical collaborative applications, safety modules implement these collaborative modes to modulate the robot's speed based on its proximity to the operator, thereby mitigating potential hazards. Conventional path planning and replanning algorithms often fall short in these contexts, as they prioritize the shortest path without considering the possible safety-related slowdowns. Consequently, researchers have devised human-aware motion planners to diminish idle times caused by safety interventions, optimizing robot movements to minimize safety-imposed speed limitations [38, 90, 122]. These methodologies bolster productivity by circumventing situations that might otherwise substantially impede the robot's performance due to frequent safety interventions, with both *proactive* and *reactive* approaches.

Proactive planners compute offline trajectories that minimize the expected interference between the operator and the robot. They rely on a model of the human's behavior and on the minimization of collision risk

[15], execution time [38], or human discomfort [109]. While these approaches are robust to the operator's presence duration and frequency of access, they become considerably less reliable when the operator's occupancy model is inaccurate or unavailable, potentially leading to unexpected interference.

Reactive planners focus on real-time robot trajectory modifications. This involves dynamic speed adjustments based on the distance from the operator [12, 108, 110, 185], as well as the ability to alter the path dynamically [66, 122, 163]. Reducing speed works well when there are short and infrequent interventions in the workspace but becomes inefficient with frequent and close collaboration, resulting in safety stops whenever the human is nearby. An emerging trend is to combine both strategies, employing path modifications during execution to avoid collisions and scaling the trajectory for safety [44, 122, 164]. Nevertheless, it is important to note that replanning algorithms usually speed up computation by prioritizing path length without considering the subsequent speed scaling, leading to further slowdowns due to the proximity of the paths found to the operator.

To overcome the limitations arising from the unpredictability of unstructured tasks in both approaches, a promising strategy to motion planning in collaborative operations is blending reactivity with proactivity. This involves integrating a reactive motion replanning algorithm with a human-aware cost function to proactively minimize interference. Cooperation efficiency is greatly improved by optimizing the trajectory for both travel times and operator-induced safety slowdowns. While previous studies are limited to offline planning when accounting for human influence [38, 90], extending this approach to real-time planning presents a challenge that requires further investigation.

5.1.1 Contributions

We aim to bridge the gap between human-aware proactive and reactive methods. Our approach integrates a safety-aware cost function based on ISO/TS 15066 [70] into a fast replanning algorithm, allowing the robot's path to be changed in runtime in a manner that strictly adheres to safety standards. The primary objective is to minimize the execution time of the trajectory, taking into account the estimated speed limitations. The proposed approach effectively addresses the limitations of proactive approaches, allowing online adaptation of the plan to accommodate unexpected movements of the operator. It also improves the efficiency of reactive approaches, going beyond simple collision-free path replanning and actively seeking safety-aware solutions. For this purpose, we draw inspiration from [38] to introduce a safety-aware cost function based on ISO/TS 15066 that can be easily integrated into the replanning algorithm. We refer to our modified replanning algorithm as **H**uman-**A**ware **M**ulti-p**A**th **R**eplanning **S**trategy (MARSHA), which extends the capabilities of MARS (Chapter 4) to the new, computationally expensive cost function, retaining high responsiveness. We validate MARSHA in simulations and real-world tests. Our experimental campaign demonstrates that the proposed method outperforms reactive and proactive methods and draws conclusions on the suitability of different planning approaches to the scenarios at hand.

A video of the algorithm is also available [1].

5.2 Related works on human-aware planning

Human-aware motion planners enable robots to move while considering the presence of humans. Previous works focused on optimizing paths by considering the human-robot distance, the human field of view, and

[1] Video: https://youtu.be/2WiNfq9vNcQ

comfort [109, 152]. Other approaches avoid high-occupancy areas by acting on the planner cost function [60, 159, 190]. For example, [60] biases the sampling to regions situated far from the current human state [159], [190] utilized STOMP [72] to minimize a cost function penalizing previously occupied regions while maximizing the human-robot distance, and [15] deformed the trajectory based on a repulsion field associated with checkpoints on the human skeleton. [13] accounts for collision risk, social norms, and crowded areas. [71] and [75] focused on reducing execution time in handover tasks without considering collisions with the environment. Some studies also evaluated human factors and demonstrated enhancements in work fluency and operator satisfaction when a human-aware motion planner is adopted [7, 91].

Recent advancements have focused on integrating safety within motion planning. Approaches like [38] and [90] employ a cost function to assess paths by estimating the speed scaling experienced by the robot during traversal. Additionally, [43] utilizes spatio-temporal human occupancy maps to devise robot trajectories that anticipate human movements. These techniques eliminate the need for fine-tuning cost function weights and directly incorporate safety-related speed limitations in motion generation, minimizing an estimate of the robot's execution time in accordance with SSM [38, 43] and PFL [90] guidelines.

However, it is important to note that these approaches perform offline planning based on a model describing human behavior during robot motion. Consequently, they face challenges in adapting to variations in expected human behavior and lack real-time adaptability.

5.3 Preliminaries

As stated in Definition 2, solving the optimal path planning problem entails determining a path $\sigma^* : [0,1] \rightarrow \mathcal{C}_{\text{free}}$ that minimizes a designated

cost function c:

$$\sigma^* = \underset{\sigma \in \Sigma}{\operatorname{argmin}} \{c(\sigma) | \sigma(0) = q_{\text{start}}, \sigma(1) = q_{\text{goal}}\} \tag{5.1}$$

$c(\sigma)$ usually measures the length of σ. Discretizing σ into $\mathcal{M}$ waypoints q, we have:

$$c(\sigma) = \sum_{i=1}^{\mathcal{M}-1} ||q_{i+1} - q_i||_2 \tag{5.2}$$

In this work, we aim to evaluate paths based on an estimate of the robot slowdown during execution. One option is to use the estimated path execution time t_{est} as a cost function, considering the human state $\mathcal{H}$, as in [38]:

$$c(\sigma) = t_{\text{est}}(\sigma, \mathcal{H}) \tag{5.3}$$

The discretization of σ allows us to express Equation (5.3) as follows:

$$c(\sigma) = \sum_{i=1}^{\mathcal{M}-1} t_{\text{nom},i} \lambda_i \left(q_i, q_{i+1}, \mathcal{H}\right) \tag{5.4}$$

where $t_{\text{nom},i}$ is the expected execution time in case the robot is not slowed down, and λ_i is the average time-dilatation factor (see Section 5.4.1) that measures the effect of the human on the execution time of connection $\overline{q_i q_{i+1}}$ of path σ:

$$\lambda_i = \frac{t_{\text{est},i}(\mathcal{H})}{t_{\text{nom},i}} \geq 1 \tag{5.5}$$

However, as detailed in Section 2.1, a prevalent strategy for addressing motion planning problems involves initially planning a path and subsequently calculating its time parameterisation. Consequently, $t_{\text{nom},i}$ is typically determined *a posteriori* utilizing methods such as TOPP [128]. Nonetheless, it can be underestimated by considering the minimum time required to cross the connection $\overline{q_i q_{i+1}}$, assuming that at least one joint

operates at its maximum speed. We adopt this approximation from [38] to decouple the cost function from the robot's velocity, which is beneficial for planners not accounting for velocities and accelerations. It is noteworthy that underestimating the time interval results in an overestimation of velocity in favor of safety. With $\dot{q}_{\max}$ the maximum joint speed vector, the minimum time required to travel $\overline{q_i q_{i+1}}$ is given by the maximum of the component-wise ratio $(q_{i+1} - q_i) \oslash \dot{q}_{\max}$; thus, Equation (5.4) becomes:

$$c(\sigma) = \sum_{i=1}^{\mathcal{M}-1} \left\| (q_{i+1} - q_i) \oslash \dot{q}_{\max} \right\|_{\infty} \lambda_i \left(q_i, q_{i+1}, \mathcal{H} \right) \tag{5.6}$$

Note that Equation (5.6) considers constant velocity along connection $\overline{q_i q_{i+1}}$, so the following equations hold:

$$\lambda_i = \frac{t_{\text{est},i}(\mathcal{H})}{t_{\text{nom},i}} = \frac{v_{\text{est},i}(\mathcal{H})}{v_{\text{nom},i}} = \frac{\dot{q}_{\text{est},i}(\mathcal{H})}{\dot{q}_{\text{nom, i}}} \geq 1 \tag{5.7}$$

where $v_{\text{est},i}(\mathcal{H})$ and $\dot{q}_{\text{est},i}(\mathcal{H})$ are the estimated Cartesian and joint velocities of the robot along connection $\overline{q_i q_{i+1}}$, which depend on the human state $\mathcal{H}$, while $v_{\text{nom},i}$ and $\dot{q}_{\text{nom},i}$ are the nominal ones, obtained considering the maximum velocity of at least one joint along $\overline{q_i q_{i+1}}$.

Optimizing a path with this cost function involves finding a solution that minimizes the estimated execution time, taking into account the human state. However, [38] does not provide an *admissible informed set* for the cost function of Equation (5.6). Including an admissible informed set in a sampling-based path planner speeds up the convergence rate by discarding areas that do not contain the optimal solution. For this reason, we approximate Equation (5.6) and derive an admissible informed set that can enhance planning.

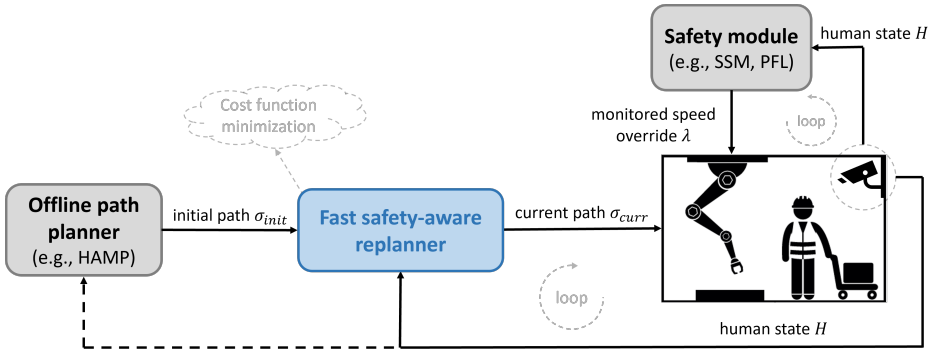


Figure 5.1: System overview. The offline path planner computes an initial path and the replanner updates the solution to minimize a safety-aware cost function when the vision system detects changes in the human state. A safety module slows down the robot based on ISO/TS 15066.

5.4 Proposed approach

The proposed approach operates within the system shown in Figure 5.1, adhering to a standard paradigm for human-robot cooperation applications. This paradigm consists of an *offline path planner* responsible for calculating the initial path, and a *fast safety-aware replanner* online updating the trajectory based on information about the human state. The offline planner can also proactively compute a trajectory that reduces the activation of safety rules. A safety module based on the ISO/TS 15066 [70] is employed to ensure safe cooperation by slowing down the robot based on the human state. Its implementation is detailed in Section 4.3.3.2.

The replanner modifies the current path whenever the human state changes, aiming to minimize the intervention required by the safety module, thereby enhancing the overall efficiency of cooperation. This goal is achieved during the replanning procedure by minimizing a safety-aware cost function, which estimates the trajectory execution time while considering the slowdown commanded by the safety module. By optimizing this cost function, the reactive planner can find an efficient solution re-

specting the safety constraints, leading to smoother and more productive human-robot cooperation.

The next subsections define the proposed safety-aware cost function, derive the admissible informed set necessary to speed up planning, and describe the overall replanning method.

5.4.1 The safety-aware cost function

Our proposal involves the design of a cost function that provides an approximation to Equation (5.6). The main objective of this cost function is to minimize the execution time while considering the human state. Simultaneously, in Section 5.4.2 we ensure that an admissible informed set remains available and can be directly sampled to accelerate the planning process. The proposed cost function is defined as follows:

$$c(\sigma) = \sum_{i=1}^{\mathcal{M}-1} \left\| (q_{i+1} - q_i) \oslash \dot{q}_{\max} \right\|_2 \lambda_i \left(q_i, q_{i+1}, \mathcal{H} \right) \tag{5.8}$$

Here, the cost function calculates the length of the path segments, weighting each joint by its maximum speed. Additionally, it incorporates the penalty term λ to discourage paths that reduce collaboration efficiency.

It is worth noting that Equation (5.8) serves as an upper limit compared to Equation (5.6). In fact, Equation (5.8) uses the Euclidean norm instead of the infinity norm and $\forall v \in \mathbb{R}^n$, $\|v\|_2 \geq \|v\|_\infty$. Thus, we have:

$$\begin{aligned} &\sum_{i=1}^{w-1} \left\| (q_{i+1} - q_i) \oslash \dot{q}_{\max} \right\|_2 \lambda_i \left(q_i, q_{i+1}, \mathcal{H} \right) \geq \\ &\sum_{i=1}^{w-1} \left\| (q_{i+1} - q_i) \oslash \dot{q}_{\max} \right\|_\infty \lambda_i \left(q_i, q_{i+1}, \mathcal{H} \right) \end{aligned} \tag{5.9}$$

Therefore, by minimizing the proposed cost function, we decrease the upper bound of Equation (5.6) and consequently reduce the estimated

execution time of the path.

The safety requirements of ISO/TS 15066 can be translated into a human-robot relative speed limit:

$$v_{rh} \leq v_{max} \tag{5.10}$$

Here, v_{rh} is the robot's velocity towards the human and v_{max} is calculated as in Section 2.4. The computation of v_{rh} follows the same procedure as outlined in Section 4.3.3.2 for the safety module, which is briefly summarized below.

Let $\mathcal{H} = \{h_1, h_2, \ldots, h_m\}$ represent a set of m points of interest for the human state, and $\mathcal{R} = \{r_1, r_2, \ldots, r_p\}$ denote a set of p points of interest on the robot's structure. Here, $h_k, r_j \in \mathbb{R}^3$ denote the Cartesian positions of the key-points on the human skeleton and robot structure, respectively.

Each point r_j is linked to the robot's configuration q through the forward kinematic function $r_j = \text{fk}_j(q)$. Additionally, u_{r_j,h_k} signifies the unit vector from r_j to h_k, and $\dot{h}_k$, $\dot{r}_j$ represent the velocities of h_k and r_j, respectively. The velocity v_{r_j,h_k} of the robot point r_j towards the human point h_k is computed as:

$$v_{r_j,h_k} = (\dot{r}_j - \dot{h}_k) \cdot u_{r_j,h_k} = (J_j(q)\dot{q} - \dot{h}_k) \cdot u_{r_j,h_k} \tag{5.11}$$

where $J_j(q)$ denotes the Jacobian of r_j.

As stated in Section 5.3, robot velocity is overestimated by considering that at least one joint is moving at its maximum speed. Thus, for connection $\overline{q_i q_{i+1}}$ we have:

$$u_i = \frac{(q_{i+1} - q_i)}{||q_{i+1} - q_i||_2}, \quad \dot{q}_i = \left(\min_t \left| \frac{\dot{q}_{\max,t}}{u_{i,t}} \right| \right) u_i \tag{5.12}$$

where $\dot{q}_{\max,t}$ and $u_{i,t}$ are the t-th component of vectors $\dot{q}_{\max}$ and u_i, respectively. Therefore, the safety slowdown $\lambda(q)$ at robot's configuration

q can be computed as follows:

$$\lambda(q) = \max_{\substack{k=1,\dots,m \\ j=1,\dots,p}} \left(\frac{v_{r_j,h_k}}{v_{max}}, 1 \right) \geq 1 \tag{5.13}$$

To elucidate, the deceleration denoted as s_{safety} in Equation (4.14) of Section 4.3.3.2 and $\lambda(q)$ embody a parallel concept. However, s_{safety} is confined to the range $[0, 1]$, whereas $\lambda(q) \geq 1$.

Equation (5.13) computes $\lambda(q)$ in a single configuration q. However, in Equation (5.8), λ_i represents the estimated average scaling factor experienced by the robot when traversing the connection $\overline{q_i q_{i+1}}$. To obtain λ_i, we divide $\overline{q_i q_{i+1}}$ into $\mathcal{Z}$ equally-spaced configurations q_z, compute $\lambda_z = \lambda(q_z)$ for $z = 1, \dots, \mathcal{Z}$ and finally:

$$\lambda_i = \frac{1}{\mathcal{Z}} \sum_{z=1}^{\mathcal{Z}} \lambda_z. \tag{5.14}$$

Note that the calculation of λ_i can be parallelized.

By substituting Equation (2.21) in Equation (5.13) to compute $\lambda(q)$, the cost function (5.8) penalizes paths failing to meet the SSM requirements. Alternatively, utilizing Equation (2.22) enables the penalization of paths that do not satisfy the PFL requirements.

5.4.2 The admissible informed set

Admissible informed sets are a powerful tool for sampling-based path planning algorithms because they speed up the search for a solution and can significantly impact performance during online replanning. The idea is to discard regions of the search space that surely do not contain the optimal solution.

Given a positive cost function, the cost $f(q)$ of the optimal path from q_{start} to q_{goal} constrained to pass through $q \in \mathcal{C}_{\text{free}}$ is equal to the cost of

the optimal path from q_{start} to q, plus the cost of the optimal path from q to q_{goal}. Since $f(\cdot)$ is generally unknown, an *admissible heuristic* $\tilde{f}(\cdot)$ is used as an estimate. An admissible heuristic is a function that never overestimates the real cost $f(q)$. In the case where the cost is the path length, the Euclidean norm represents an admissible heuristic because the cost of a path from q_{start} to q_{goal} passing through q cannot be lower than:

$$\tilde{f}(q) = \|q - q_{\text{start}}\|_2 + \|q_{\text{goal}} - q\|_2 \leq f(q) \tag{5.15}$$

It follows that all points outside of the following prolate hyperspheroid:

$$\mathcal{I} = \{q \in \mathcal{C}_{\text{free}} \mid ||q - q_{\text{start}}||_2 + ||q_{\text{goal}} - q||_2 < c\} \tag{5.16}$$

will surely not improve the current solution [49] (here, c denotes the cost of the current solution). Such a set is called an *admissible informed set*.

Considering that $\lambda \geq 1$ in Equation (5.8), it is possible to define an admissible heuristic setting $\lambda = 1$:

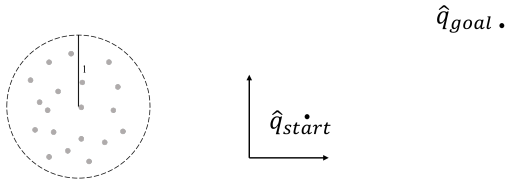
$$\tilde{f}(q) = \|(q - q_{\text{start}}) \oslash \dot{q}_{\max}\|_2 + \|(q_{\text{goal}} - q) \oslash \dot{q}_{\max}\|_2 \tag{5.17}$$

from which the following admissible informed set derives:

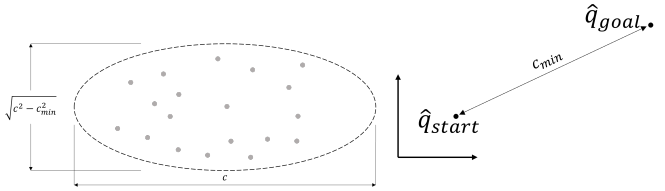
$$\begin{aligned} \mathcal{I} = \{q \in \mathcal{C}_{\text{free}} \mid & ||(q - q_{\text{start}}) \oslash \dot{q}_{\max}||_2 \\ & + ||(q_{\text{goal}} - q) \oslash \dot{q}_{\max}||_2 < c\} \end{aligned} \tag{5.18}$$

Note that Equation (5.17) differs from Equation (5.15) on the fact that Equation (5.17) weighs each joint by the inverse of its maximum speed. If we define $\hat{q} = \hat{g}(q) = q \oslash \dot{q}_{\max}$ and $\hat{\mathcal{C}}_{\text{free}} = \hat{g}(\mathcal{C}_{\text{free}})$, Equation (5.18) can be written as:

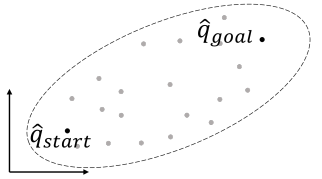
$$\hat{\mathcal{I}} = \{\hat{q} \in \hat{\mathcal{C}}_{\text{free}} \mid ||\hat{q} - \hat{q}_{\text{start}}||_2 + ||\hat{q}_{\text{goal}} - \hat{q}||_2 < c\} \tag{5.19}$$



1. Uniform random sampling of a unit ball.



2. Stretching of the unit ball to form an hyperellipsoid.



3. Rototranslation of the hyperellipsoid to align the foci with $\hat{q}_{start}$ and $\hat{q}_{goal}$.

Figure 5.2: Illustration of the admissible informed set sampling procedure.

Hence, to obtain a direct sample from the informed set described in Equation (5.18), we can follow a similar sampling procedure employed by [49] to sample the set of Equation (5.19), illustrated in Figure 5.2 and which, in summary, involves:

- Uniform random sampling of a unit ball.
- Extending the unit ball to form a hyperellipsoid with a transverse diameter equal to c and conjugate ones equal to $\sqrt{c^2 - c^2_{\text{min}}}$, where c_{min} represents the minimum cost between $\hat{q}_{\text{start}}$ and $\hat{q}_{\text{goal}}$.
- Rototranslating the hyperellipsoid to align the foci with $\hat{q}_{\text{start}}$ and $\hat{q}_{\text{goal}}$.

After sampling $\hat{q}$ from the set represented by Equation (5.19), we can then compute q by applying the inverse function $\hat{g}^{-1}$.

5.4.3 Human-Aware Multi-pAth Replanning Strategy

This section presents **H**uman-**A**ware **M**ulti-p**A**th **R**eplanning **S**trategy (MARSHA), a variant of MARS, presented in Chapter 4, to quickly replan with the new presented cost function.

The modifications made to MARS and the framework are highlighted in red in Algorithm 8 and 9, respectively. Specifically, we addressed the computational burden associated with the computation of the cost function, which can be significantly heavier compared to the simple Euclidean norm used in MARS. To tackle this issue, we adopted a lazy approach that minimizes the need for frequent evaluations of the cost function (line 10 of Algorithm 9). During the process of subtree expansion, our algorithm employs a strategy that evaluates the cost of new connections only when necessary. For instance, if the selected path planning algorithm does not use a rewiring procedure (*e.g.*, RRT [93]), the cost of connections is assessed only after the subtree has successfully connected

Algorithm 8 The modified replanning architecture for HRC.

Thread *Trajectory execution*

1: $t \leftarrow t + \Delta t$
2: state $\leftarrow$ sampleTrajectory(τ, t)
3: sendToController(state)

Thread *Collision check*

4: $\mathcal{H} \leftarrow$ updateHumanState()
5: $\sigma_{\text{subpath}} \leftarrow \sigma_{\text{current}}[q_{\text{current}}, q_{\text{goal}}]$
6: $\mathcal{P} \leftarrow \mathcal{S} \cup \sigma_{\text{subpath}}$
7: **for** $\sigma_j \in \mathcal{P}$ **do**
8: free, $q_{\text{before}}, q_{\text{after}} \leftarrow$ checkCollisions(σ_j)
9: **if** free **then**
10: $c_{\sigma_j} \leftarrow c(\sigma_j, \mathcal{H})$
11: **else**
12: $c_{\sigma_j} \leftarrow +\infty$

Thread *Path replanning*

13: $q_{\text{current}} \leftarrow$ projectOnPath(state, σ_{current})
14: $\sigma_{\text{replanned}}$, solved $\leftarrow$ replan$(\sigma_{\text{current}}, q_{\text{current}}, \mathcal{S}, \text{max_time}, \mathcal{H})$
15: **if** solved **then**
16: $\sigma_{\text{current}} \leftarrow \sigma_{\text{replanned}}$
17: $\tau \leftarrow$ computeTrajectory$(\sigma_{\text{current}})$
18: **else if** distance from obstacle $\leq$ minimum allowed distance **then**
19: sendRobotStop()

Algorithm 9 MARSHA: high-level description

1: Define $\mathcal{P}$ as the set of available paths
2: Define $\mathcal{Q}_1$ as the queue of the nodes of σ_{current} between q_{current} and the obstacle
3: Sort $\mathcal{Q}_1$ by some criterion
4: **for each** node $q_n \in \mathcal{Q}_1$ **do**
5: Insert the nodes of each $\sigma \in P$ into a queue $\mathcal{Q}_2$
6: Sort $\mathcal{Q}_2$ by some criterion
7: **for each** $q_j \in \mathcal{Q}_2$ **do**
8: Define the informed set on the best cost up to now
9: Get the subtree rooted in q_n from the informed set
10: Grow the subtree in the informed set to reach q_j with lazy cost evaluation
11: Update $\mathcal{Q}_1$ if a solution was found
12: Get the best path from the graph

to its target node. In this way, we limit the calculation of the cost function only to the connections that belong to the solution.

It is important to note that the evaluation of the cost function requires the updated human state, $\mathcal{H}$, as an input (line 14 of Algorithm 8). For example, $\mathcal{H}$ can be provided by a human tracking system. The *Collision Check Thread*, in addition to verifying collisions along the available paths, updates $\mathcal{H}$ (line 4) and calculates the cost of the connections for the paths accordingly (line 10). This approach ensures that MARSHA does not waste time evaluating the cost of connections of the current path and of the set of available paths, but only focuses on the connections belonging to the new solutions that have not yet been evaluated during the same replanning iteration.

Figure 5.3 illustrates the distinct behaviors expected from MARS and MARSHA. Specifically, MARS focuses on finding the shortest path that connects the start to the goal (dashed black line), disregarding any effects the SSM module may have on the path itself. Consequently, part of the path goes through the yellow region, triggering a limited safety intervention (*e.g.,* limited slowdown), and the red region, which requires a strong safety intervention (*e.g.,* robot halts). On the other hand, MARSHA fa-

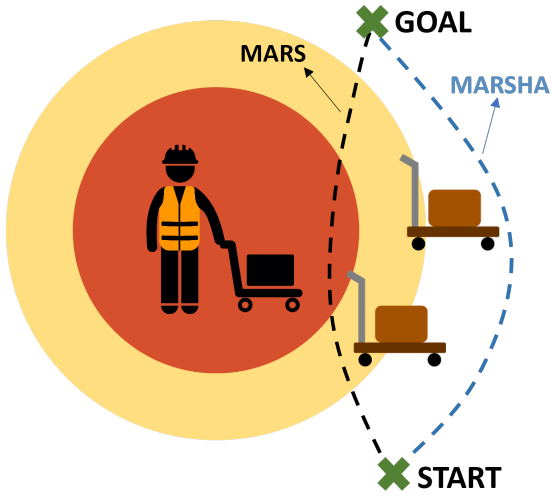


Figure 5.3: MARSHA prioritizes the fastest path (dashed blue line), taking into account that entering the yellow and red areas leads to speed reductions and a complete stop, respectively. On the other hand, MARS gives precedence to the shortest path (dashed black line).

vors a longer path that can be traversed more quickly (dashed blue line).

5.5 Experiments

This section assesses MARSHA's performance in scenarios involving partially unstructured collaborative tasks and various collaboration regimes.

We conducted a comparative analysis between MARSHA and both reactive and proactive approaches. Specifically, we evaluated two reactive approaches alongside MARSHA. The first approach, dSSM (dynamic SSM 2D [12]), continually monitors both human and robot statuses, determining the maximum allowable robot speed based on Equation (2.21). If the robot's speed exceeds the maximum allowed by safety protocols, it is scaled by a factor λ. This approach is effectively implemented by the safety module, as outlined in Section 4.3.3.2. Concerning Figure 5.1, there is no replanning block. The second reactive approach used in the experiments combines MARS with dSSM. In the first case, the robot slows down according to the safety criterion defined by Equation (2.21). In the second case, the robot has the additional capability to replan its path. The dSSM approach is expected to block the robot as long as the operator is nearby, while MARS modifies the path to find collision-free solutions, but at the cost of significantly reduced speed. It is worth noting that MARS modifies the path only when encountering an obstacle or identifying a shorter solution. Instead, MARSHA predicts the effect of the safety module on the solutions found and aims to minimize the traversing time, considering the need for speed slowdowns. MARS and MARSHA are assigned a maximum calculation time of 200 ms. As per Algorithm 8, the robot controller follows the actual trajectory at a high rate (*e.g.,* 500 Hz) as well as the safety module, while trajectories may changes every 200 ms or less, provided the replanning algorithm successfully identifies a suitable solution within the allocated timeframe.

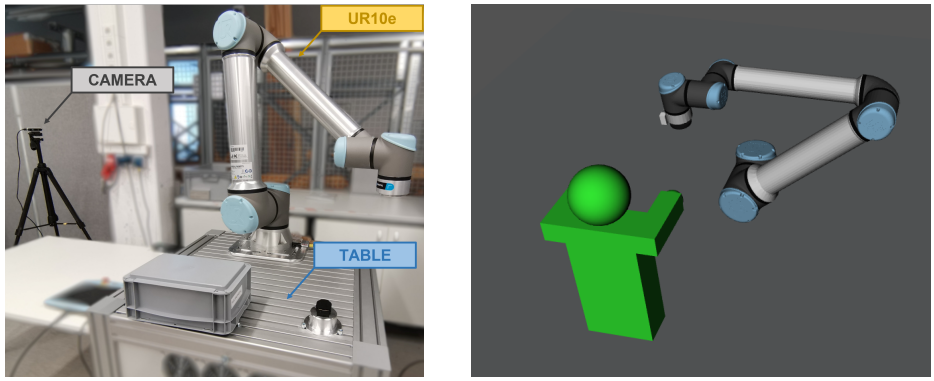
Regarding proactive approaches, MARSHA was compared to HAMP [38], a safety-aware path planning algorithm, and RRT* [76], where the human expected volume is considered as an obstacle. Both approaches were combined with dSSM in runtime. HAMP seeks to find a time-optimal solution that minimizes Equation (5.6). Conversely, RRT* aims to identify the shortest path. HAMP is expected to generate solutions that require minimal or no intervention from the safety module as long as the operator remains within the space expected during the planning phase. However, if the operator deviates from it, the performance of HAMP deteriorates. RRT* is expected to find shorter paths but at the expense of triggering the safety module frequently. Referring to Figure 5.1, both tests with HAMP and RRT* do not implement any replanning block.

We evaluated the following metrics:

- *Execution time*: The time taken to complete the task. Results are normalized relative to the minimum execution time, *i.e.*, the one computed without human presence.

- *Average scaling factor*: The average speed override (between 0 and 100) forced by the safety module to ensure safety. A lower scaling factor indicates a greater level of intervention by the safety module in reducing the robot's speed.

We conducted simulated and real-world experiments to have extensive simulation results followed by validation in a real-world collaborative cell. The setup consists of a Universal Robot UR10e (6-degree-of-freedom collaborative robot) and a shared table. In the real-world experiments (Figure 5.4a), an Intel RealSense D435 tracks the human. We compute the human skeleton bounding boxes and feed them into the SSM computation. In simulations, we replicate the human body and movements using a translating mannequin (see Figure 5.4b) with added uni-



(a) Real robotic cell. **(b)** Simulated environment.

Figure 5.4: Experimental setups.

form noise with a maximum amplitude of 3 cm. The locations of the mannequin's head, torso, arms, and hands are used for the SSM computation. For safety reasons, we limit the robot's speed to 30% of its maximum value during real experiments.

It is worth emphasizing that while the proposed cost function could be potentially utilized in both SSM and PFL applications, as explained in Section 5.4.1, our focus in this study has been primarily on the former. Table 5.1 summarizes the parameters used by the SSM (refer to Equation (2.21)).

The robot control pipeline follows the one in Figure 5.1. MARSHA and the reactive methods are implemented in the *fast safety-aware replanner* module, while the proactive methods are implemented in the *offline path planner* module. The implementation of the human skeleton tracking system, safety module, and replanner mirrors the methodology detailed in Section 4.3.3.

The tasks consist of a common shared workspace scenario, where the robot's objective is to move from an initial configuration to a designated target configuration to execute tasks like pick&place operations. Concur-

Table 5.1: Parameters of the SSM safety module used for testing.

$T_r\ (s)$	$a_s\ (m/s^2)$	$C\ (m)$	$v_h\ (m/s)$
0.15	2.5	0.25	1.6

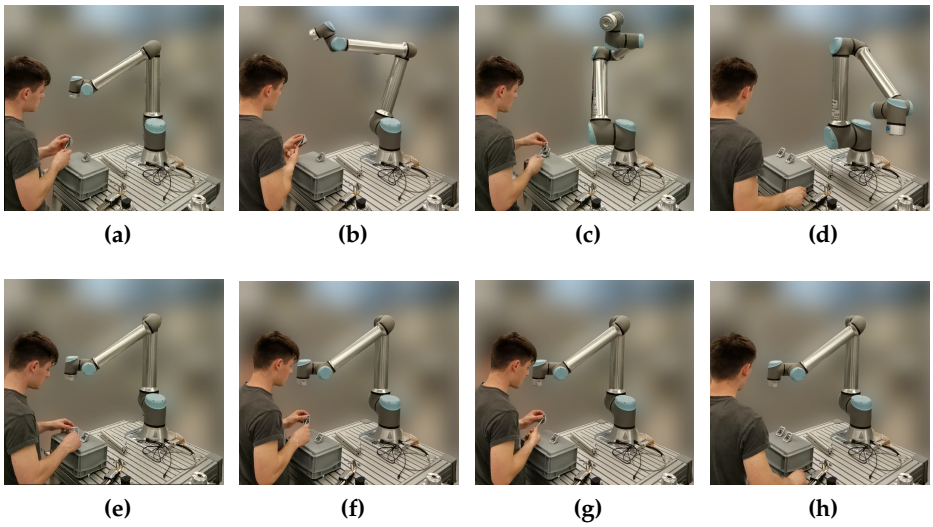


Figure 5.5: (a)-(d) MARSHA enables the robot to dynamically adjust its path in response to the operator's approach. (e)-(h) dSSM halts the robot as long as the operator is in close proximity.

rently, an operator approaches and works on the same table, sharing the workspace with the robot.

Beyond demonstrating the effectiveness of the proposed approach, our aim is to offer clear insights into scenarios where the performance enhancement justifies the added complexity in the system, as well as cases where it may not. By doing so, we can make informed decisions on whether the benefits outweigh the costs of implementing the approach.

5.5.1 Comparison with reactive approaches

The experiments primarily varied based on the duration of the operator's presence at the workbench. This variation enabled us to investigate different interaction scenarios and assess their impact on the performance of the proposed approach. In particular, the tests were carried out as follows. The robot plans a path from start to goal, while the operator is still far away. The trajectory initially calculated corresponds to the fastest one in the absence of the operator. Two additional paths are computed for MARS and MARSHA. As the robot begins to move, the operator approaches it to work on the shared table. Depending on the algorithm being used at the time, the robot simply slows down or replans its path to avoid the person or to minimize the estimated execution time considering the need for safety slowdowns. After a pre-established time interval, the operator leaves the work table. We will distinguish the tests in three cases: *short*, *medium*, and *long*, which are distinguished by the time during which the operator is close to the robot, which is 5, 10 and 20 seconds, respectively. For each type of test and algorithm, 50 simulations and 20 experiments on the real cell were performed. Fig 5.5 shows snapshots of MARSHA and dSSM running during the experiments.

Figure 5.6 depicts the outcomes of the *short*, *medium*, and *long* tests conducted in the simulation environment. It should be noted that a lower execution time and a higher average scaling factor indicate superior algorithm performance. Upon analysis, it becomes evident that MARSHA outperforms both MARS and dSSM in all tested scenarios. MARSHA exhibits the capability to generate path plans that significantly reduce activations of the safety module, enabling the robot to reach its goal more expeditiously. This behavior remains consistent across all three test variations, regardless of the operator's duration of interaction with the workbench. Conversely, MARS and dSSM exhibit poor performance, particularly when the operator remains near the robot for an extended period.

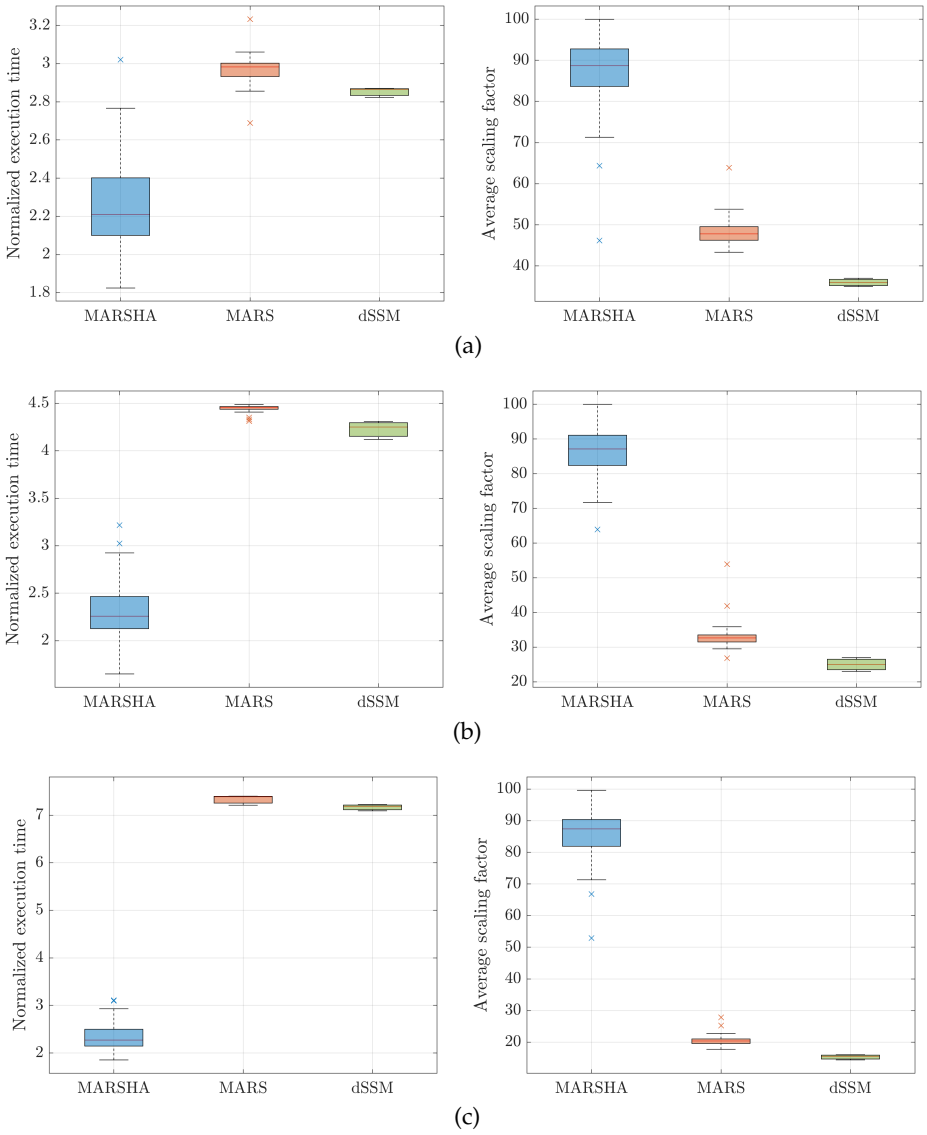
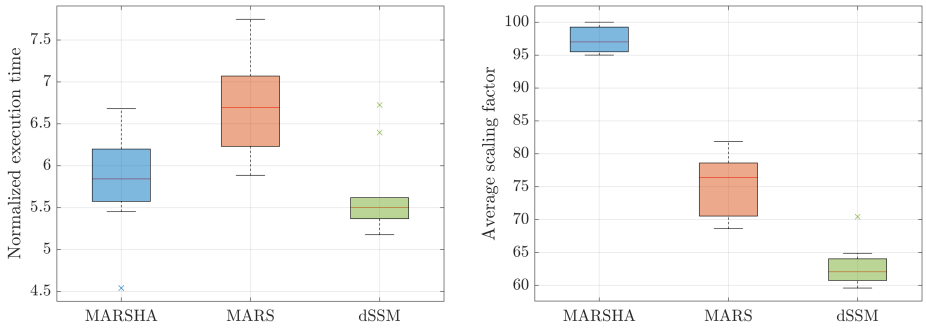


Figure 5.6: Comparison with reactive approaches in simulation. (a) *short* interaction (5 seconds), (b) *medium* interaction (10 seconds), (c) *long* interaction (20 seconds).

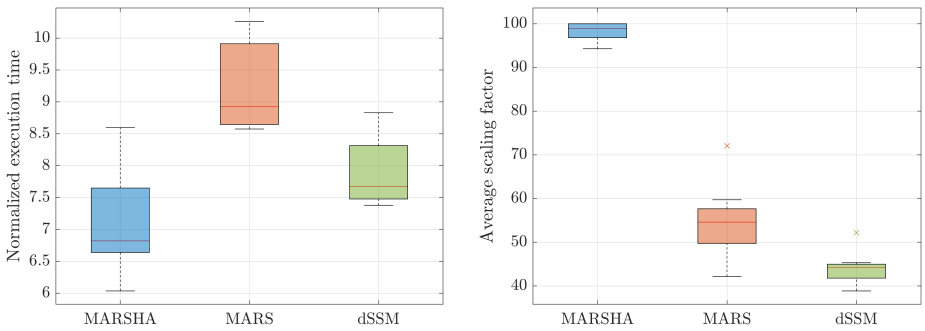
As anticipated, dSSM restricts the robot's movement for the entire duration of the operator's presence nearby. Moreover, MARS's replanning feature enhances safety by ensuring collision-free paths but compromises the achieved performance. Since MARS continuously finds and optimizes new paths in terms of length, the solutions frequently bring the robot near the operator, eliciting strong responses from the safety module. Based on these evaluations, the performance gap between MARSHA and MARS/dSSM is expected to widen as the operator's time spent at the robot increases.

The findings from the simulations are further validated through experiments conducted on the physical robot, as illustrated in Figure 5.7. However, the performance disparity between the algorithms narrows. This can be attributed to the fact that, as anticipated, the robot operates at only 30% of its full capacity during real tests. Consequently, the distance covered by the robot before the operator departs is significantly reduced. As a result, the robot often does not reach the portion of the path replanned by MARSHA before the operator moves away. In this way, the advantages offered by MARSHA are not fully utilized, sometimes leading to a comparable performance with dSSM. This effect becomes more pronounced as the robot's maximum speed decreases and the duration of the operator's presence at the workstation diminishes.

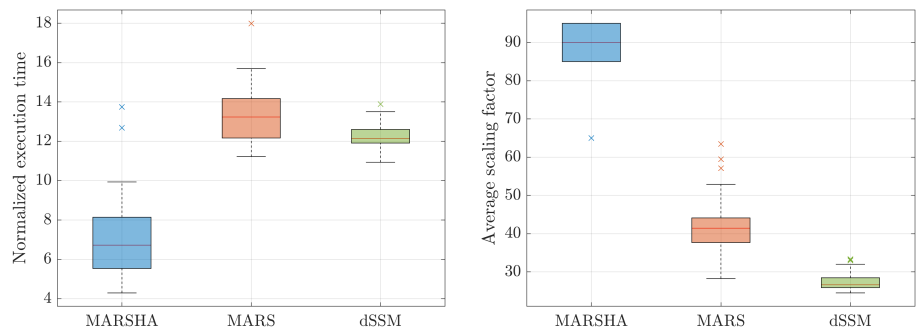
To confirm this hypothesis, we conducted simulated *short* tests with constant scaling at both 30% and 60%, then combined with the scaling provided by the SSM module. The results, as depicted in Figure 5.8, clearly indicate that reducing the robot's speed corresponds to a smaller performance gap between the algorithms. Consequently, when the robot's speed is significantly restricted, and the operator's interventions in the cell are brief and infrequent, MARSHA performs as dSSM and it does not provide notable advantages over dSSM alone. Additionally, other factors, such as the positioning of the operator and the timing of entry



(a)



(b)



(c)

Figure 5.7: Real experiments with reactive approaches. (a) *short* interaction (5 seconds); (b) *medium* interaction (10 seconds); (c) *long* interaction (20 seconds).

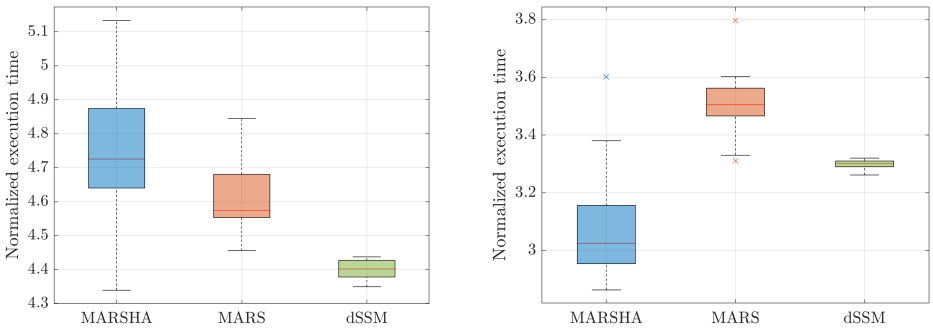


Figure 5.8: The maximum robot speed is scaled to 30% (left) and 60% (right).

into the cell, may partly influence the results, as they are not always the same during real tests.

However, the results demonstrate that MARSHA excels in identifying faster executable paths by considering the impact of the safety module on the robot during execution. Notably, as the duration of the operator's presence in the shared space increases, MARSHA exhibits enhanced performance compared to MARS/dSSM.

5.5.2 Comparison with proactive approaches

MARSHA's performance has been compared to proactive approaches, aiming to analyze the advantages it offers over them. Unlike proactive approaches, MARSHA updates its plan in real time to adapt to unexpected operator movements. It is important to note that MARSHA is not meant to replace proactive approaches but rather complement them as a reactive approach. In the tests, we evaluate MARSHA's performance when the initial path is calculated using a proactive approach and when it is calculated without considering safety protocols. The tests were conducted as follows: the robot's objective is to move from a starting configuration to a goal configuration while the operator works in the shared

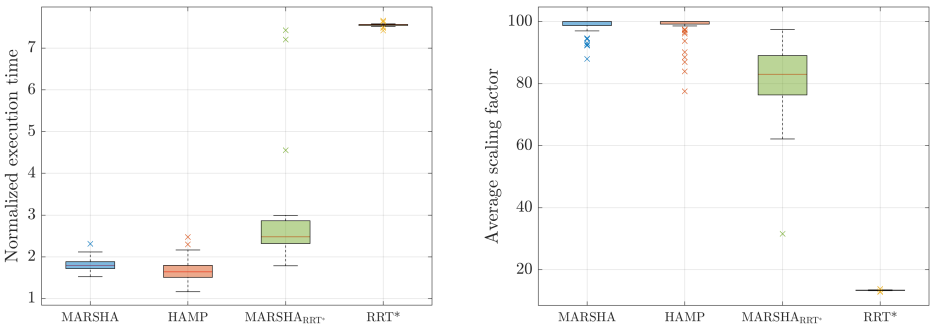


Figure 5.9: Test with proactive approaches in simulation.

space for 20 seconds. The operator is already near the work table when the initial path is calculated, allowing proactive approaches to factor in the operator's position during the planning phase.

Each algorithm (HAMP [38], MARSHA, and RRT* [76]) was tested with 50 simulations and 20 trials in the real cell.

HAMP plans a path that minimizes execution time by considering the operator's position during the planning phase, while RRT* finds the shortest path while avoiding collisions with the person. Since RRT* does not proactively account for safety, it is expected to have longer execution times. MARSHA was tested with initial paths calculated using both HAMP and RRT*. Comparing these two methodologies helps us understand if MARSHA's performance is heavily influenced by the initial path calculation. The results are illustrated in Figure 5.9 and 5.10. Note that MARSHA$_{RRT^*}$ refers to MARSHA with the initial path computed using RRT*.

The conducted tests yield intriguing conclusions. When the initial path of MARSHA is safety-aware, HAMP and MARSHA exhibit comparable execution times. Remarkably, in some cases, MARSHA even outperforms HAMP by efficiently accommodating deviations caused by the operator. Additionally, the fixed 20 seconds planning time of HAMP can

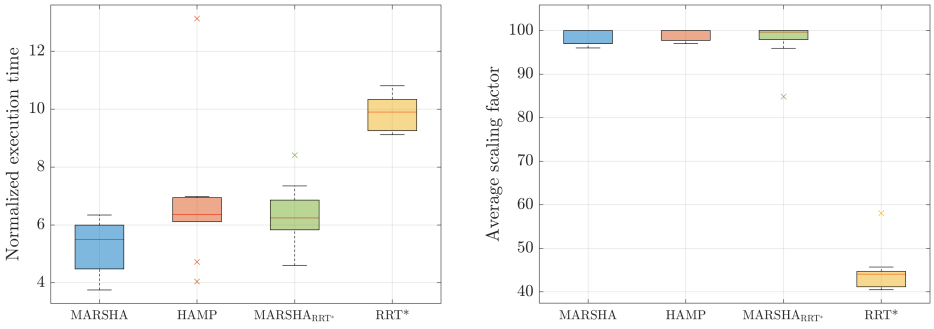


Figure 5.10: Test with proactive approaches on the real robot.

lead to suboptimal solutions, whereas MARSHA's ability to improve the plan during execution results in time savings. As expected, RRT* performs poorly due to its path's excessive proximity to the operator, which results in significant slowdowns of the shortest path. However, an intriguing result emerges when the initial path of MARSHA is generated using RRT*. Surprisingly, even without considering human-awareness in the initial path, MARSHA's performance remains relatively unaffected compared to using a human-aware path. This is a significant advantage, as MARSHA can provide performance similar to a human-aware planner using a simpler and faster generic initial path calculation (taking just a few seconds instead of tens of seconds). This outcome solidifies MARSHA's position as a hybrid control strategy, skillfully combining elements from proactive and reactive planners, and showcasing unique characteristics from both categories.

Note that in real experiments, we observe a narrowing gap between RRT* and the other algorithms as well. This phenomenon is closely related to the constant scaling at 30%. When the operator is near the table, the robot halts, rendering the scaling effect insignificant during those 20 seconds of stoppage. However, when the operator is farther away, allowing the robot to move freely, it follows the shortest path, and the

constant scaling affects a short trajectory. On the contrary, the other algorithms consistently maintain a certain distance from the operator, leading to longer travels at a reduced speed of 30%. Consequently, the impact of the constant scaling becomes more pronounced for these algorithms.

MARSHA demonstrates superior adaptability compared to HAMP when the operator's behavior deviates from expectations, such as sudden movements or walking away. HAMP's trajectory can experience significant slowdowns in the event of unexpected operator motion, as it lacks real-time adaptation capabilities. When the operator moves away from the designated area, HAMP continues to follow the initially calculated trajectory, assuming the operator is still in the shared workspace. This leads to a longer trajectory, as it is computed prior to the operator's departure to ensure minimal safety intervention. In contrast, MARSHA swiftly adjusts its plan by directing the robot along the quickest path to effectively accommodate the operator's movement.

5.5.3 Effect of the SSM parameters on the performance

This section focuses on analyzing the impact of SSM parameters (described by Equation (2.21)) on MARSHA's performance and how the performance gap with MARS/dSSM changes accordingly. Table 5.1 summarizes the parameters employed in the experiments conducted in Sections 5.5.1 and 5.5.2. By modifying these parameters, the degree to which the safety module intervenes to decelerate the robot varies. Certain parameter combinations prompt immediate and pronounced reactions from the safety module as soon as the operator enters the shared workspace, while other combinations allow for delayed intervention. It is important to note that these parameters are typically determined through a safety analysis conducted on the specific robotic cell, taking into account factors such as system reaction times (T_r) and the robot's maximum Cartesian acceleration (a_s), among others. This section aims to examine algorithm perfor-

mance as these parameters are varied, thus extending the applicability of the findings to robotic cells with different characteristics.

Two values were chosen for each parameter to ensure comprehensive coverage, resulting in a total of 16 combinations. For each combination, the *long* test was repeated 50 times for each algorithm. Table 5.2 presents the selected parameter values, including those representing a more reactive cell and those simulating a less reactive cell. Additionally, two speed values were considered for the human operator: 0 m/s, which assumes the operator is stationary at the work table, and 1.6 m/s, which aligns with the recommended value by ISO/TS 15066 when measuring the human walking speed is not feasible. Figure 5.11 shows that varying sets of parameters have distinct effects on the performance of MARSHA.

MARSHA consistently outperforms MARS/dSSM in the majority of cases and demonstrates comparable performance in the worst-case scenarios. With conservative implementations, MARSHA's behavior aligns with that of the SSM because the minimum allowed human-robot distance prevents MARSHA from generating solutions that do not trigger strong safety interventions. Less conservative implementations allow for a smaller safety distance. Consequently the collision-free paths furnished by MARS become sufficient to achieve commendable performance levels. In the remaining cases, MARSHA provides shorter execution times.

5.5.4 Discussion of the results

We compared MARSHA with reactive and proactive approaches in partially unstructured contexts and with different types of interaction.

In contrast to reactive approaches (MARS and dSSM), MARSHA finds solutions that minimize interventions from the safety module and allows to reach the goal faster. This behavior is evident when the operator stays in the cell for longer periods, while it is less significant for sporadic and brief interventions. This result highlights the importance of the applica-

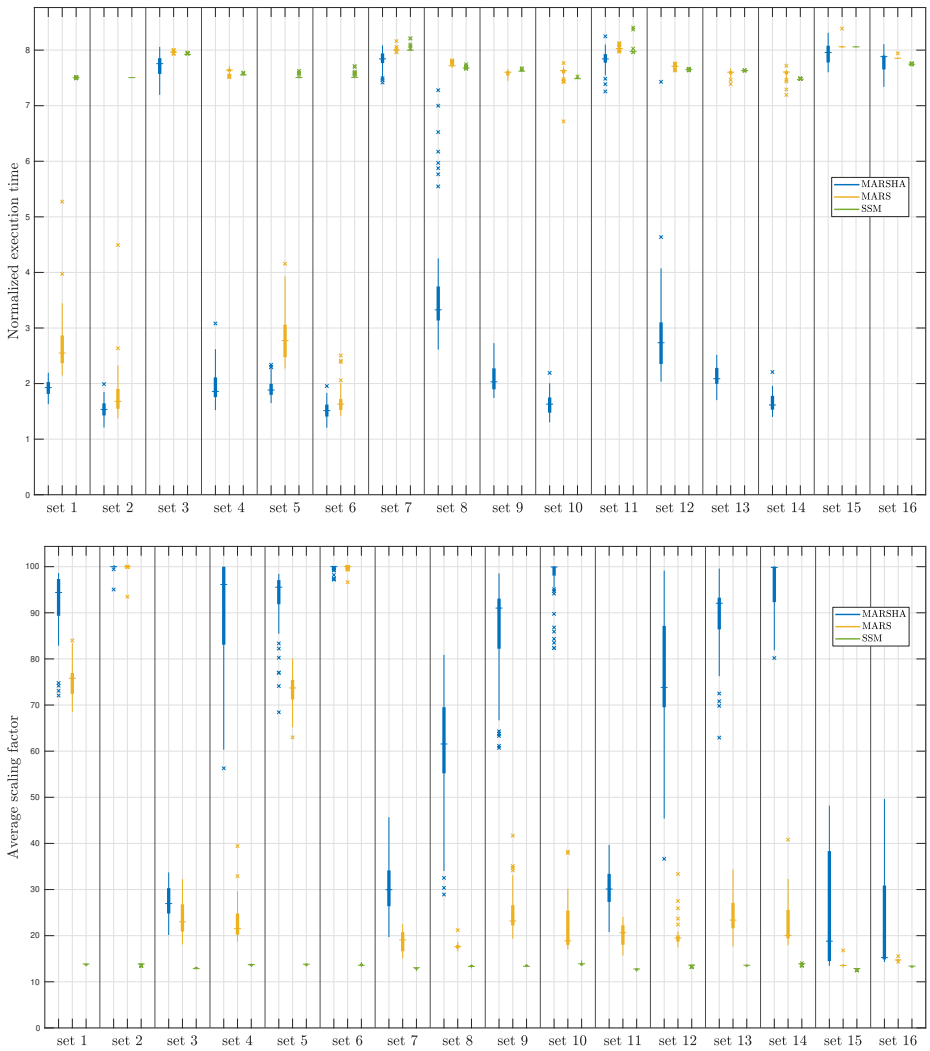


Figure 5.11: *Long* test with the SSM parameter sets in Table 5.2.

Table 5.2: Different parameter sets for the SSM safety module used to evaluate MARSHA.

set	1	2	3	4	5	6	7	8
$C\ (m)$	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10
$T_r\ (s)$	0.15	0.15	0.15	0.15	0.30	0.30	0.30	0.30
$v_h\ (m/s)$	0.00	0.00	1.60	1.60	0.00	0.00	1.60	1.60
$a_s\ (m/s^2)$	0.10	2.50	0.10	2.50	0.10	2.50	0.10	2.50

set	9	10	11	12	13	14	15	16
$C\ (m)$	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30
$T_r\ (s)$	0.15	0.15	0.15	0.15	0.30	0.30	0.30	0.30
$v_h\ (m/s)$	0.00	0.00	1.60	1.60	0.00	0.00	1.60	1.60
$a_s\ (m/s^2)$	0.10	2.50	0.10	2.50	0.10	2.50	0.10	2.50

tions' features in the selection of the most effective planning approach. Roughly speaking, the advantages of a safety-aware planner will be justified mainly in applications with a long interaction times.

We extensively evaluated MARSHA's performance under different configurations of the SSM module. When employing conservative settings, MARSHA's behavior closely mirrors that of the SSM. However, with less conservative settings, MARS proves capable of achieving commendable performance levels on its own. In the remaining scenarios, MARSHA consistently achieves shorter execution times.

MARSHA outperformed offline planners (HAMP and RRT*) because it can improve solutions in real-time and adjust to unforeseen operator movements. For instance, if the operator leaves the cell, MARSHA falls back to the fastest path, while offline methods keep following the precomputed path. MARSHA performs well even when its initial path is not human-aware thanks to the online optimization. This confirms its hybrid nature, combining the advantages of proactive and reactive approaches.

5.6 Summary

This chapter presented MARSHA, an innovative motion replanning algorithm enhancing human-robot cooperation applications. MARSHA performs real-time optimization of the robot's trajectory execution by employing a safety-aware cost function. This function estimates the time required to traverse a path, factoring in potential safety-related slowdowns. The result is a notable reduction in safety stops and slowdowns caused by the proximity of humans to the robot. To maintain a high level of responsiveness in the replanning algorithm, we presented an admissible informed set for the proposed safety-aware cost function.

Compared to reactive approaches, MARSHA significantly reduces the intervention of the safety module by anticipating safety slowdowns and dynamically adjusting the trajectory. These benefits are particularly pronounced in scenarios involving extended periods of human-robot interaction. Instead, in situations involving rapid and sporadic human interventions, MARSHA performs comparably to a standalone SSM implementation. We thoroughly evaluated the performance of MARSHA across a diverse range of safety-related parameters. Additionally, unlike proactive techniques, MARSHA exhibits the capability to accommodate shifts in the operator's position, thereby averting further safety interventions.

In conclusion, MARSHA is a promising solution for enhancing human-robot collaboration by effectively optimizing robot trajectories while considering safety aspects and remaining flexible to accommodate changing operator positions. Its hybrid approach opens up new possibilities for seamless human-robot cooperation across various application domains.

CHAPTER 6

OpenMORE

With the spread of robots in unstructured, dynamic environments, the topic of path replanning has gained importance in the robotics community. Although the number of replanning strategies has significantly increased, there is a lack of agreed-upon libraries and tools, making the use, development, and benchmarking of new algorithms arduous. This chapter introduces OpenMORE, a new open-source ROS-based C++ library for sampling-based path replanning algorithms. The library builds a framework that allows for continuous replanning and collision checking of the traversed path during the execution of the robot trajectory. Users can solve replanning tasks exploiting the already available algorithms and can easily integrate new ones, leveraging the library to manage the entire execution.

This chapter is based on the work presented in [161].

6.1 Introduction

With the advances in industrial, social, and exploration robotics, robots are increasingly required to work in dynamic environments (*e.g.*, sharing their workspace with humans). For this reason, it is necessary for the robot to deploy a reactive behavior, enabling it to react promptly to changes occurring in the work environment. For example, a robot sharing the workspace with an operator will often be obstructed by the latter. To avoid collisions and reduce downtime, the robot should be able to promptly change its path without stopping.

Definition 3 (Section 2.2) underscores the heightened complexity of the path replanning problem in comparison to the standard path planning problem (as defined in Definition 1). In path planning, a path is usually computed considering the environment as static, and subsequently it is used to generate a trajectory for the robot controllers. There is no active planning during the execution of the path, which may lead to invalidity in the presence of dynamic obstacles. In contrast, a path replanning problem requires continuous monitoring of the scene during trajectory execution and repeated online planning to adapt the solution found to new environmental information. Thus, in order to properly perform its function, a path replanner requires an architecture that is responsible for monitoring the scene, triggering the replanning, and contextually executing the robot's trajectory.

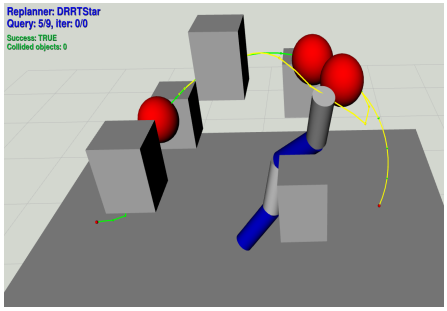
There are several libraries dedicated to path planning [30, 131, 155], but the same cannot be said for path replanning. In particular, OMPL [155] has become the standard library for motion planning. It implements many state-of-the-art planners and provides integration with other software tools such as *MoveIt!* [69]. OMPL deals only with static environments; that is, it does not implement any architectures to concurrently handle scene tracking, current path adaptation and trajectory execution, necessary to solve the path replanning problem in dynamic en-

vironments. Recently, *MoveIt!* [69] has introduced a hierarchical architecture (*Hybrid Planning*) for adapting a motion plan online. It follows a classic hierarchical approach with a global planner that computes an initial trajectory and a local planner that interpolates it and makes local changes. When the latter fails, the robot stops and the global planner is queried again. This hierarchical architecture is a common strategy inspired by mobile robots, yet it prioritizes local changes and invokes the replanner only when the local planner is stuck. However, our interest is on continuous and global replanning and execution, for which, to the author's best knowledge, no open-source libraries are available online. This situation significantly complicates the process of using, testing and comparing new algorithms against the state-of-the-art, as implementing and evaluating an algorithm is both time-consuming and inherently influenced by the choices made during the implementation phase.

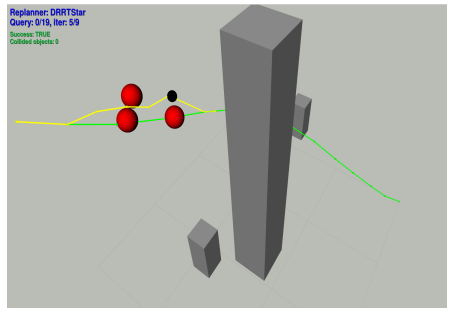
6.1.1 Contributions

The goal of this chapter is to present OpenMORE, an open-source C++ library based on ROS [138] that implements an entire framework to handle smooth and continuous sampling-based path replanning, scene tracking and trajectory execution concurrently. One can easily use the already available algorithms (the list is expanding) and integrate new ones quickly, without spending time on building the entire necessary architecture. The library develops abstract classes that limit the effort required to develop and implement new replanners.

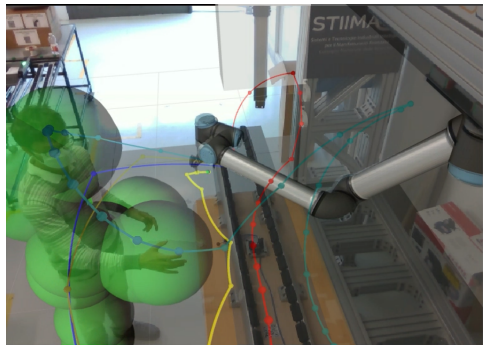
OpenMORE can be used to solve replanning problems in dynamic environments, where the initially calculated path can be invalidated by moving obstacles. For example, in Figure 6.1c it was used to allow humans and robots to share their workspace in a human-robot cooperation application by modifying online the robot path according to the operator movements [164]. The library was also used in [163] to benchmark



(a) Simulated manipulator



(b) Simulated 3D robot



(c) Replanning in human-robot collaboration

Figure 6.1: Examples of application of OpenMORE. In simulated environments, (a)-(b), red spheres are unexpected obstacles, while green and yellow lines denotes the initial and the current path, respectively. In the real robotic cell (c), the yellow path indicates the robot's current one, while the other coloured paths are employed by the replanning algorithm to compute a new solution.

several replanning algorithms in different scenarios (Figures 6.1a-6.1b).

6.2 Library overview

The purpose of OpenMORE is to provide a tool that makes it easier to use, test, implement, and benchmark sampling-based path replanning algorithms for research and education. For this reason, its development took into consideration the following aspects:

- *Efficiency*: the library is developed in C++ to obtain a reliable and fast tool, features needed to get quick reactions from the robot;
- *Easy to use*: available replanners can be used easily, little code is needed to launch an execution;
- *Easy development*: new sampling-based path replanning algorithms can be easily integrated into the framework, using the guidelines and tools provided by the abstract classes;
- *Easy integration*: the library is integrated with ROS, allowing for easy interaction with additional software components (*e.g.*, replanning and safety speed monitoring runs together in [164])

The library primarily focuses on the development of a framework to handle replanning, scene monitoring and trajectory execution simultaneously. In addition, the library has features that aid in debugging and benchmarking, such as path visualization, simulation of random moving obstacles, and collection of useful data. OpenMORE has a few dependencies, many of which are simple internally implemented ROS packages that support the architecture. The main one is *graph_core*, a library that defines the classes needed for a path planning problem and some sampling-based algorithms to solve it. The only external dependencies

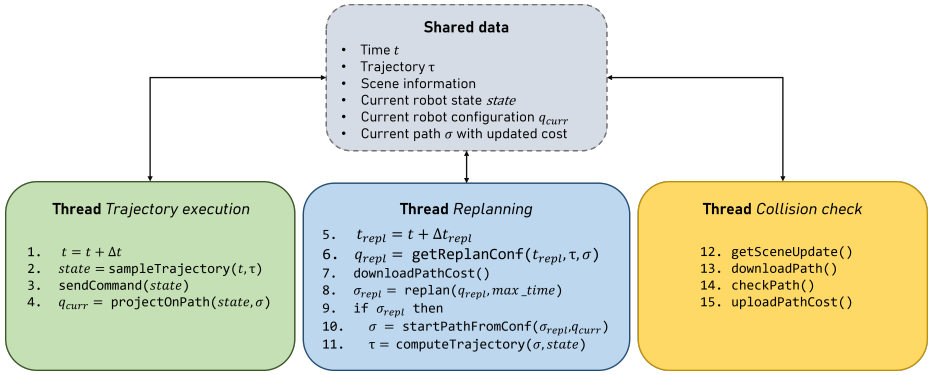


Figure 6.2: Conceptual overview of the `replanner_manager`

are Eigen [54] and *MoveIt!* [69], which is used to track the planning scene and for collision checking.

6.3 Core concepts

The `replanner` and the `replanner_manager` are the key components of the library. The `replanner` serves as an abstraction for the path replanning algorithm, while the `replanner_manager` defines the entire architectural framework.

6.3.1 The `replanner`

The `replanner` is the actor that executes the replanning algorithm when commanded by the `replanner_manager`. `replanner` provides a base class in which to define the replanning algorithm, but its implementation is up to the user. It defines the structure that a replanner must have to interact with the `replanner_manager` and requires that the `replan` function be implemented. The information the `replanner` needs are:

- the configuration of the robot: a new path will be searched starting from this configuration;

- the current path and/or tree: they are usually used to find quickly a new solution by replanning algorithms;
- the maximum replanning time: to get responsive behavior from the robot, the algorithm should provide a solution in a short amount of time;
- the path planning problem solver: the algorithm used to find a path from the current configuration to the goal.

For example, DRRT [40] is a replanning algorithm which deletes the invalid part of the tree and rebuilds it starting from the goal and working its way back to the robot's configuration.

It therefore needs to know the tree, the current configuration of the robot, and a solver (in this case RRT [93]) to rebuild the tree.

6.3.2 The `replanner_manager`

Figure 6.2 shows a conceptual overview of the architecture of the `replanner_manager`, which implements the framework outlined in Chapter 3, briefly summarized as follows:

- *Trajectory Execution Thread*: micro-interpolates the robot's current trajectory to send the new command to the robot controller at a high rate.
- *Collision Checking Thread*: updates the scene information and checks for the collisions along the path in execution. It considers the part of the current path from the current robot configuration q_{curr} to the goal.
- *Path Replanning Thread*: invokes the `replanner` to execute the replanning algorithm to find a path when the current one is blocked

or to optimize the current solution. If the replanner finds a new path, then it computes a trajectory on it.

These three components are roughly in charge of execution, collection of obstacle data, and replanning, respectively. The *Trajectory Execution Thread* is usually the one that runs the fastest because it is the interface between the framework and the controller.

The trajectory τ undergoes interpolation at line 2 of Figure 6.2, resulting in the state variable (*state*) which encapsulates joint positions (q_{state}), velocities, and accelerations. Subsequently, the interpolated state is projected onto the nominal path σ at line 4. Denoted as $s_{\text{abs}} \in [0, 1]$ the curvilinear abscissa of σ, the projection q_{curr} is achieved by solving the following optimization problem:

$$\begin{aligned} s_{\text{curr}} &= \underset{s_{\text{abs}} \in [0,1]}{\operatorname{argmin}} \left\| q_{\text{state}} - \sigma(s_{\text{abs}}) \right\| \\ q_{\text{curr}} &= \sigma(s_{\text{curr}}) \end{aligned} \tag{6.1}$$

Algorithm 10 provides a straightforward way to address the problem of Equation (6.1). This projection step is essential due to the inherent nature of time parameterization algorithms, which can introduce minor discrepancies from the path computed by the path planner (*e.g.*, due to blending radii). In contrast, the replanning algorithm relies on nodes that strictly adhere to the path or tree.

In parallel, the *Collision Check Thread* takes a snapshot of the scene (line 12), updates the path cost based on this scene (line 14), and shares the information with the *Path Replanning Thread* thread (line 15), which will rely on this snapshot and the current path cost calculated to search for a new solution. Finally, the replanning algorithm defined by the `replanner` determines how the new path is search and/or improved (line 8).

Information is exchanged between threads by means of shared data

Algorithm 10 projectOnPath

Input: the path σ, the point to project q_{state}, the abscissa stepsize Δabs
Ensure: the projected point q_{curr}

1: $d \leftarrow \infty$
2: $\text{abs} \leftarrow 0$
3: **while** $\text{abs} \leq 1$ **do**
4: $\quad d' \leftarrow \|q_{\text{state}} - \sigma(\text{abs})\|$
5: $\quad$ **if** $d' < d$ **then**
6: $\quad\quad d \leftarrow d'$
7: $\quad\quad q_{\text{curr}} \leftarrow \sigma(\text{abs})$
8: $\quad \text{abs} \leftarrow \text{abs} + \Delta\text{abs}$ ▷ E.g. $\Delta\text{abs} = 0.01$
9: **return** q_{curr}

(grey box in Figure 6.2). Specifically, each thread owns a copy of the current path, which is updated whenever it is updated in the shared data. This allows threads (especially the replanning and collision check threads) to carry out their tasks without interblocking. There is therefore a mechanism for downloading and uploading the shared information at the beginning and at the end of the threads. For example, the collision checking thread updates the local copy of the path, if necessary, at the beginning of each iteration (line 13). After that, it uploads the calculated cost and scene information into the shared path with the other threads (line 15). The replanning thread therefore downloads this information at the beginning of the iteration (line 7) and at the end uploads the new calculated path and trajectory.

Because the replanner has a planning latency while the robot is moving, it is necessary to consider the displacement between the expected and the real position at the end of the replanner query. So, to obtain a smoother transition from the current path to the new one, the `replanner` replans by considering as if the robot were in a state later in time than the current one (line 8). To do that, the replanning configuration q_{repl} is

obtained by sampling the trajectory at time instant $t + \Delta t_{\text{repl}}$ and then projecting the state on the current path. This is done by `getReplanConf` at line 6. Consequently, when a new path is found, it must be adapted to have the most recent q_{curr} as starting configuration (`startPathFromConf` function at line 10). Δt_{repl} should be a value equal to or slightly greater than the maximum time given to the replanner to find a solution. The trajectory is then computed considering the robot's current conditions, represented by *state* at line 11. The choice of the time parameterisation algorithm does not alter either the architecture or the replanning algorithm. As outlined in Section 2.1, we leverage state-of-the-art techniques that adhere to the kinematic and dynamic constraints of the robot, such as [89].

The diagram in Figure 6.3 illustrates the workflow of OpenMORE integrated into the ROS framework [138]. Scene updates are transmitted from *MoveIt!* [69] via the `moveit_msgs::GetPlanningScene` service. Note that to incorporate new information about obstacles, publishing them within the *MoveIt!* scene is imperative (*e.g.*, as done in Section 4.3.3.1). Simultaneously, the `replanner_manager` executes the robot's current trajectory while concurrently planning a new one. To transmit commands to the robot's low-level controller, the replanner can either be integrated into a ROS Controller [24] or send it the new commands using a message of type `sensor_msgs::JointState`.

6.3.3 Development of a new replanner

The implementation and integration into the framework of a new replanning algorithm can be summarized with the following pipeline:

1. implementation of a `replanner`'s child class and its `replan` function;

2. implementation of a `replanner_manager`'s child class that contains

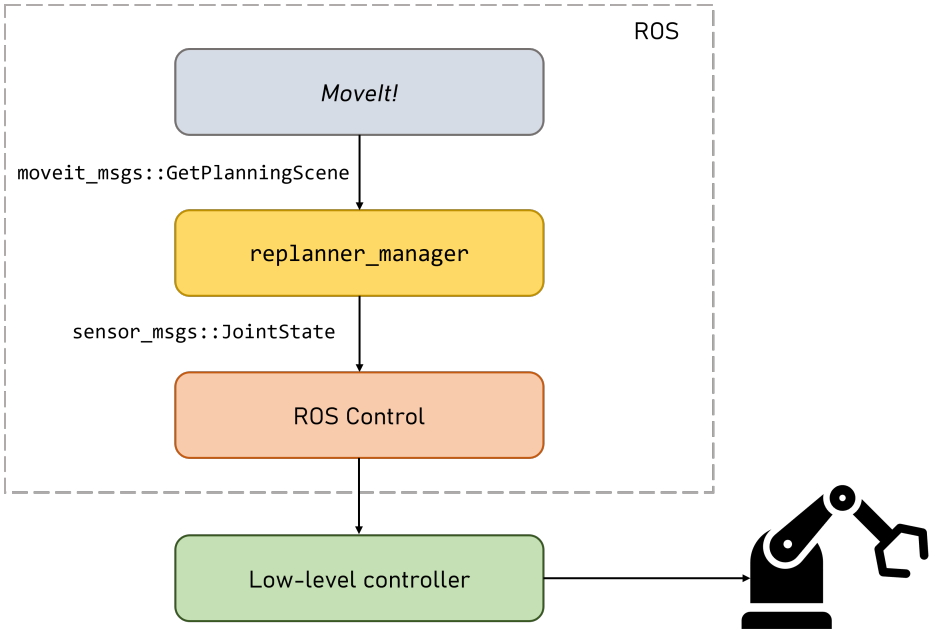


Figure 6.3: OpenMORE pipeline. Scene updates are received from *MoveIt!* and new commands sent to the robot low-level controller through ROS Control.

and initializes the `replanner` object;

3. definition of the triggering condition of the `replanner` (*e.g.*, replan when there are collisions along the current path or continuously to refine the current solution);

4. implementation of the `startPathFromConf` function: it defines how to set q_{curr} as starting configuration of the path. This could happen simply by adding a node in the tree and extrapolating the path to the goal, but in general it strongly depends on the implemented replanning algorithm, so the definition of the function is left to the user. Note that this function should not be computationally expensive compared to the actual replanning (line 8);

Clearly this represents the minimum requirement to be able to integrate a new algorithm within the framework. However, in a replanning context there are many important aspects, *i.e.* the generation of the trajectory and the management of moving obstacles for simulations. Default functions are already provided and may be suitable for many cases, but the user has the possibility to customize and override them.

6.3.4 Debugging and visualization

The library has useful features for debugging and benchmarking. Among these we find the real time visualization of the current path on RViz and the random appearance of unexpected obstacles, as shown in Figure 6.1a and 6.1b. A data collection thread is also available to collect data for benchmarking. In particular, the latter carries out statistical analyzes on the replanning time during an entire execution, the length of the path actually traversed, the success or failure of the execution and possibly the number of obstacles with which a collision has occurred. All of this, together with different levels of verbosity, can be turned on or off by parameters.

6.3.5 Available replanners

The library currently implements some sampling-based path replanning algorithms, such as MARS [163], DRRT [40], Multi Parallel RRT [157], Anytime DRRT [42], and [27], which have been successfully used for benchmarking in [163]. One can take advantage of the implementation provided in the library to have a starting point for new developments.

```
// 1) Init ROS node handle nh, MoveIt move group and planning scene
// 2) Compute the initial path
Eigen::VectorXd lb(6),ub(6),start(6),goal(6);
lb.setConstant(-1.0); ub.setConstant(1.0);//joints bounds
start.setRandom(); goal.setRandom();

MetricsPtr metrics = std::make_shared<Metrics>();
InformedSamplerPtr sampler =
std::make_shared<InformedSampler>(start,goal,lb,ub);
CollisionCheckerPtr checker =
std::make_shared<MoveitCollisionChecker>(pln_scn,groupname);

RRTStarPtr solver =
std::make_shared<RRTStar>(metrics,checker,sampler);
PathPtr path; solver->computePath(start,goal,nh,path);

// 3) Create and start replanner manager
ReplannerManagerDRRT replanner_manager(path,solver,nh);
replanner_manager.start();
```

Figure 6.4: Simple C++ code to solve a path replanning problem using OpenMORE.

6.4 Example applications

Figure 6.4 shows a simple C++ example to solve a path replanning problem. First, create a ROS node handle and initilize the *MoveIt!* move group. Then, set the robot joints bounds, start and goal configurations, and initialize the metric (which is used to evaluate the path's cost, *i.e.* Euclidean norm), the sampler and the collision checker. Note that the latter needs a planning scene object to evaluate collisions. Now you can use a path planning solver to find the initial path. Finally, create the replanner manager and start the trajectory execution with replanning. More details and tutorials can be found at [160].

Figure 6.1c shows an application of the framework in a human-robot collaboration context. A vision system identifies the human's skeleton and publishes bounding spheres to the *MoveIt!* scene. Based on this, the replanner is able to alter the path to avoid the operator. Thanks to the integration with ROS, it was possible to make the replanner interact easily

with an ISO/TS15066-based speed scaling module launched on a different node. The replanner was inserted into a controller in the ROS Control framework [24] which read the commands from the `replanner_manager` and sent them to the robot. Figure 6.1a and 6.1b are snapshots from a simulated environment, where robots are asked to move from a start configuration to a goal configuration while some unexpected obstacles appear.

6.5 Summary

This chapter presented OpenMORE, an open-source C++ ROS-based library for easy use and deployment of sampling-based path replanning algorithms. The library offers a comprehensive framework for managing trajectory execution with continuous replanning and collision checking of the current path. Additionally, it provides a range of valuable tools for algorithm usage, implementation, debugging, and benchmarking. The primary objective of this library is to offer researchers and students an off-the-shelf architecture to use and develop online motion planning algorithms. The library is a work-in-progress actively under development and can be accessed at [160]. Ongoing efforts involve the creation of extensive documentation and tutorials to enhance user-friendliness. Further developments under evaluation are the generalization of the software used for scene monitoring and the integration with OMPL. A ROS-free version of the library is being developed to allow integration with other frameworks and facilitate migration to ROS2, offering users platform flexibility.

CHAPTER 7

Conclusions

This thesis introduced a novel approach to path replanning, motivated by the growing demand for robots to move in dynamic environments, *e.g.*, in scenarios involving human-robot collaboration. While standard path planning algorithms work well in static environments, real-world situations often involve changing surroundings, necessitating rapid robot responses to prevent collisions. The existing literature offers a variety of strategies to address replanning challenges, including optimization-based, learning-based, graph-based, and sampling-based methods. Among these, the sampling-based approach distinguishes itself for its simplicity, adaptability, and proficiency in managing high-dimensional search spaces without explicit representation.

However, proposed techniques, like roadmap-based strategies, face challenges in maintaining consistent quality paths in the presence of numerous obstacles. Additionally, tree-based methods may entail computationally expensive pruning or rewiring phases, particularly in cases involving robots with many degrees of freedom, such as manipulators.

For effective human-robot collaboration in dynamic environments, rapid generation of new paths in response to operator-induced obstructions is crucial for safety and operational efficiency.

Moreover, considering that in HRC scenarios safety modules adjust the robot's speed based on proximity to humans, it becomes imperative for the replanning algorithm to account for these effects in solution generation. This allows to find safety-compliant solutions that optimize collaboration efficiency by reducing the need of slowdowns and halts for the robot.

To address these challenges, this thesis proposes the following contributions:

- **Replanning Framework:** Conventional motion planning involves the computation of a path, its temporal parametrisation, and the subsequent execution of the trajectory. However, real-world environments with dynamic obstacles require a more adaptive approach. Chapter 3 introduces an architecture that enables simultaneous trajectory execution, collision checking, and replanning. This system utilizes three concurrent threads, ensuring uninterrupted motion. Moreover, it offloads collision checking of the current path from the replanning algorithm, enhancing efficiency.

- **Novel Sampling-Based Path Replanning Algorithm:** Chapter 4 introduces MARS, an innovative sampling-based path replanning algorithm based on the exploitation of a set of pre-computed paths. Its key innovation lies in connecting the current path to the nodes from the other available paths and utilizing subpaths towards the goal. By prioritizing nodes that are closer to the current path than the goal, MARS reduces search complexity. The algorithm employs a dynamic graph structure built over iterations, benefiting from prior planning efforts. Techniques such as subtree reuse, informed sampling, and lazy collision checks expedite computations.

- **Safety-Aware Path Replanning Algorithm:** Chapter 5 delves into motion replanning within scenarios involving human-robot collaboration. Conventional replanning algorithms often neglect safety considerations, potentially resulting in paths that necessitate safety interventions. This chapter introduces a novel cost function that evaluates paths based on estimated execution time while accounting for safety-related slowdowns. Additionally, it presents an admissible informed set to expedite online solution computations. The proposed cost function is integrated into the algorithm of Chapter 4, which has been adapted to deal with the increased computational complexity. The resulting algorithm, denoted as MARSHA, enables more efficient solutions in HRC settings.

- **Tool For Developing, Testing and Using Sampling-Based Path Replanning Algorithms:** Implementing a replanning algorithm necessitates a sophisticated architecture, as outlined in Chapter 3. Unfortunately, there are no existing open-source libraries that offer such a framework, making the development, testing, and utilization of replanning algorithms more challenging. OpenMORE addresses this gap by providing the necessary architecture for managing trajectory execution alongside simultaneous replanning. Additionally, it offers implementations of various sampling-based path replanning algorithms and valuable tools to streamline the development and testing process.

The proposed algorithms were evaluated through extensive simulation campaigns and tests in real-world robotic cells. More specifically, we conducted a comparative analysis between MARS and four alternative replanning algorithms. Additionally, MARSHA was assessed alongside both proactive and reactive techniques in the context of human-robot collaboration. The findings underscored the algorithms' ability to quickly discover new solutions even in complex scenarios involving robots with

many degrees of freedom. Notably, MARS demonstrated a capacity to uncover shorter paths compared to its counterparts, whereas MARSHA contributed to reductions in the execution times of the robot's trajectory. All the tests were carried out using the proposed architecture and its implementation in OpenMORE.

Collectively, these endeavors were driven by the aspiration to advance the field of motion planning in dynamic environments by presenting pioneering solutions to the challenges of path replanning in HRC.

List of Publications

The work of this thesis is the result of several publications. In addition, collaborations have been carried out in the fields of control and robotics.

Published

- C. Tonola, M. Faroni, N. Pedrocchi, and M. Beschi. Anytime informed path re-planning and optimization for human-robot collaboration. In *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, pages 997–1002, 2021.

- C. Tonola, M. Faroni, M. Beschi, and N. Pedrocchi. Anytime informed multi-path replanning strategy for complex environments. *IEEE Access*, 11:4105–4116, 2023.

- C. Tonola, M. Beschi, M. Faroni, and N. Pedrocchi. OpenMORE: an open-source tool for sampling-based path replanning in ROS. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, 2023.

- C. Tonola, M. Faroni, N. Pedrocchi, and M. Beschi. Anytime informed path re-planning and optimization for robots in changing environments. *arXiv preprint arXiv:2103.13245* (2021).

- M. Beschi, C. Tonola, and A. Visioli. Teaching control courses online during the covid-19 pandemic: some experiences at the University of Brescia. *IFAC-PapersOnLine*, 55(17):103–108, 2022. 13th IFAC Symposium on Advances in Control Education ACE 2022.

- R. Adamini, N. Antonini, A. Borboni, S. Medici, C. Nuzzi, R. Pagani, A. Pezzaioli, and C. Tonola. User-friendly human-robot interaction based on voice commands and visual systems. In *2021 24th International Conference on Mechatronics Technology (ICMT)*, pages 1–5, 2021.

Submitted

- Tonola, C., Faroni, M., Abdolshah, S., Hamad, M. , Haddadin, S., Pedrocchi, N., Beschi, M. Reactive and safety-aware path replanning for collaborative applications. Submitted to *IEEE Transactions on Automation Science and Engineering*.

Bibliography

[1] J. Abdor-Sierra, E. Merchán-Cruz, F. Sánchez-Garfias, R. Rodríguez-Cañizo, E. Portilla-Flores, and V. Vázquez-Castillo. Particle swarm optimization for inverse kinematics solution and trajectory planning of 7-dof and 8-dof robot manipulators based on unit quaternion representation. *Journal of Applied Engineering Science*, 19(3):592–599, 2021.

[2] S. Aine and M. Likhachev. Truncated incremental search. *Artificial Intelligence*, 234:49–77, 2016.

[3] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev. Multi-Heuristic A*. *The International Journal of Robotics Research*, 35(1-3):224–243, 2016.

[4] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A Random Sampling Scheme for Path Planning. In *Robotics Research*, pages 249–264, 1996.

[5] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann. Blazepose: On-device real-time body pose tracking, 2020.

[6] R. Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[7] M. Beschi, M. Faroni, C. Copot, and N. Pedrocchi. How motion planning affects human factors in human-robot collaboration. *IFAC-PapersOnLine*, 53(5):744–749, 2020.

[8] R. Bohlin and L. Kavraki. Path planning using lazy PRM. In *Proceedings 2000 ICRA. IEEE International Conference on Robotics and Automation*, volume 1, pages 521–528, 2000.

[9] M. Branicky, S. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, volume 2, pages 1481–1487, 2001.

[10] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *International Journal of Robotic Research*, 21:1031–1052, 2002.

[11] J. Bruce and M. M. Veloso. Real-time randomized path planning for robot navigation. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 2752:288–295, 2003.

[12] C. Byner, B. Matthias, and H. Ding. Dynamic speed and separation monitoring for collaborative robot applications – concepts and performance. *Robotics and Computer-Integrated Manufacturing*, 58:239–252, 2019.

[13] K. Cai, W. Chen, C. Wang, S. Song, and M. Q.-H. Meng. Human-Aware Path Planning With Improved Virtual Doppler Method in Highly Dynamic Environments. *IEEE Transactions on Automation Science and Engineering*, 20(2):1304–1321, 2023.

[14] G. Carabin and L. Scalera. On the Trajectory Planning for Energy Efficiency in Industrial Robotic Systems. *Robotics*, 9(4), 2020.

[15] A. Casalino, D. Bazzi, A. M. Zanchettin, and P. Rocco. Optimal proactive path planning for collaborative robots in industrial contexts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6540–6546, 2019.

[16] B. Chandler and M. A. Goodrich. Online RRT* and online FMT*: Rapid replanning with dynamic cost. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6313–6318, 2017.

[17] N. Chao, Y. kuo Liu, H. Xia, M. jun Peng, and A. Ayodeji. DL-RRT* algorithm for least dose path Re-planning in dynamic radioactive environments. *Nuclear Engineering and Technology*, 51(3):825–836, 2019.

[18] J. Chase Kew, B. Ichter, M. Bandari, T.-W. E. Lee, and A. Faust. Neural collision clearance estimator for batched motion planning. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 73–89. Springer, 2020.

[19] P. Chen and Y. Hwang. SANDROS: a motion planner with performance proportional to task difficulty. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*, volume 3, pages 2346–2353, 1992.

[20] Y. Chen, Z. He, and S. Li. Horizon-based lazy optimal RRT for fast, efficient replanning in dynamic environment. *Autonomous Robots*, 43(8):2271–2292, Dec 2019.

[21] R. Cheng, K. Shankar, and J. W. Burdick. Learning an Optimal Sampling Distribution for Efficient Motion Planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7485–7492, 2020.

[22] G. Chiriatti, G. Palmieri, C. Scoccia, M. C. Palpacelli, and M. Callegari. Adaptive Obstacle Avoidance for a Class of Collaborative Robots. *Machines*, 9(6), 2021.

[23] G. Chiriatti, G. Palmieri, C. Scoccia, M. C. Palpacelli, and M. Callegari. Adaptive obstacle avoidance for a class of collaborative robots. *Machines*, 9(6), 2021.

[24] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, and E. Fernández Perdomo. ros_control: A generic and simple control framework for ROS. *The Journal of Open Source Software*, 2017.

[25] B. Cohen, M. Phillips, and M. Likhachev. Planning Single-arm Manipulations with n-Arm Robots. In *Proceedings of Robotics: Science and Systems*, July 2014.

[26] T. Combelles, C. Marmonnier, L. Proffit, and L. Jouanneau. MORRFx and Its Framework: Multi-objective Sampling Based Path Planning for Unpredictable Environments. In *2022 7th International Conference on Mechanical Engineering and Robotics Research (ICMERR)*, pages 1–6, 2022.

[27] D. Connell and H. La. Dynamic path planning and replanning for mobile robots using RRT*. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1429–1434, 2017.

[28] N. Das and M. Yip. Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, 36(4):1096–1114, 2020.

[29] J. de Gea Fernández, D. Mronga, M. Günther, T. Knobloch, M. Wirkus, M. Schröer, M. Trampler, S. Stiene, E. Kirchner, V. Barg-

sten, T. Bänziger, J. Teiwes, T. Krüger, and F. Kirchner. Multimodal sensor-based whole-body control for human–robot collaboration in industrial settings. *Robotics and Autonomous Systems*, 94:102–119, 2017.

[30] R. Diankov and J. Kuffner. OpenRAVE: A Planning Architecture for Autonomous Robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

[31] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[32] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, nov 1993.

[33] A. D. Dragan, K. C. Lee, and S. S. Srinivasa. Legibility and predictability of robot motion. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 301–308, 2013.

[34] M. Duguleana, F. G. Barbuceanu, A. Teirelbar, and G. Mogan. Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning. *Robotics and Computer-Integrated Manufacturing*, 28(2):132–146, 2012.

[35] R. Ebendt and R. Drechsler. Weighted A* search – unifying view and application. *Artificial Intelligence*, 173(14):1310–1342, 2009.

[36] M. Elbanhawi and M. Simic. Sampling-Based Robot Motion Planning: A Review. *IEEE Access*, 2:56–77, 2014.

[37] M. Faccio, I. Granata, and R. Minto. Task allocation model for human-robot collaboration with variable cobot speed. *Journal of Intelligent Manufacturing*, pages 1–14, 2023.

[38] M. Faroni, M. Beschi, and N. Pedrocchi. Safety-aware time-optimal motion planning with uncertain human state estimation. *IEEE Robotics and Automation Letters*, 7(4):12219–12226, 2022.

[39] M. Faroni, M. Beschi, N. Pedrocchi, and A. Visioli. Predictive inverse kinematics for redundant manipulators with task scaling and kinematic constraints. *IEEE Transactions on Robotics*, 35(1):278–285, 2018.

[40] D. Ferguson, N. Kalra, and A. Stentz. Replanning with RRTs. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1243–1248, 2006.

[41] D. Ferguson and A. Stentz. Anytime RRTs. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5369–5375, 2006.

[42] D. Ferguson and A. Stentz. Anytime, dynamic planning in high-dimensional search spaces. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1310–1315, 2007.

[43] J. Flowers, M. Faroni, G. Wiens, and N. Pedrocchi. Spatio-Temporal Avoidance of Predicted Occupancy in Human-Robot Collaboration, 2023.

[44] J. Flowers and G. Wiens. A Spatio-Temporal Prediction and Planning Framework for Proactive Human–Robot Collaboration. *Journal of Manufacturing Science and Engineering*, 145(12):121011, 10 2023.

[45] T. Fraichard. Trajectory planning in a dynamic workspace: a 'state-time space' approach. *Advanced Robotics*, 13(1):75–94, 1998.

[46] J. D. Gammell. *Informed anytime search for continuous planning problems*. PhD thesis, University of Toronto, 2017.

[47] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa. Informed sampling for asymptotically optimal path planning. *IEEE Transactions on Robotics*, 34(4):966–984, 2018.

[48] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa. Batch informed trees (BIT*): Informed asymptotically optimal anytime search. *International Journal of Robotics Research*, 39(5):543–567, 2020.

[49] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014.

[50] R. Gayle, K. R. Klingler, and P. G. Xavier. Lazy Reconfiguration Forest (LRF) - An Approach for Motion Planning with Multiple Tasks in Dynamic Environments. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1316–1323, 2007.

[51] R. Gayle, A. Sud, M. C. Lin, and D. Manocha. Reactive deformation roadmaps: motion planning of multiple robots in dynamic environments. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3777–3783, 2007.

[52] S. Ge and Y. Cui. Dynamic Motion Planning for Mobile Robots Using Potential Field Method. *Autonomous Robots*, 13(3):207–222, Nov 2002.

[53] R. Geraerts and M. Overmars. Reachability Analysis of Sampling Based Planners. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 404–410, 2005.

[54] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[55] J. Ha and S. Kim. Fast Replanning Multi-Heuristic A*. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7430–7435, 2021.

[56] S. Haddadin, S. Haddadin, A. Khoury, T. Rokahr, S. Parusel, R. Burgkart, A. Bicchi, and A. Albu-Schäffer. On making robots understand safety: Embedding injury knowledge into control. *The International Journal of Robotics Research*, 31(13):1578–1602, 2012.

[57] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[58] F. Hauer and P. Tsiotras. Deformable Rapidly-Exploring Random Trees. In *Robotics: Science and Systems*, 2017.

[59] K. Hauser. On responsiveness, safety, and completeness in real-time motion planning. *Autonomous Robots*, 32(1):35–48, Jan 2012.

[60] R. Hayne, R. Luo, and D. Berenson. Considering avoidance and consistency in motion planning for human-robot manipulation in a shared workspace. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3948–3954, 2016.

[61] 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 Journal of the Industrial Electronics Society*, 3:353–365, 2022.

[62] C. Hocaoglu and A. Sanderson. Evolutionary path planning using multiresolution path representation. In *Proceedings 1998 IEEE International Conference on Robotics and Automation*, volume 1, pages 318–323 vol.1, 1998.

[63] C.-M. Huang and B. Mutlu. Anticipatory robot control for efficient human-robot collaboration. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 83–90, 2016.

[64] E. Huang, M. Mukadam, Z. Liu, and B. Boots. Motion planning with graph-based trajectories and gaussian process inference. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5591–5598, 2017.

[65] J. Huh and D. D. Lee. Efficient Sampling With Q-Learning to Guide Rapidly Exploring Random Trees. *IEEE Robotics and Automation Letters*, 3(4):3868–3875, 2018.

[66] M. Hüppi, L. Bartolomei, R. Mascaro, and M. Chli. T-PRM: Temporal Probabilistic Roadmap for Path Planning in Dynamic Environments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10320–10327, 2022.

[67] B. Ichter, J. Harrison, and M. Pavone. Learning Sampling Distributions for Robot Motion Planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094, 2018.

[68] B. Ichter and M. Pavone. Robot Motion Planning in Learned Latent Spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019.

[69] S. Ioan A. and C. Sachin. MoveIt. https://moveit.ros.org.

[70] ISO Central Secretary. ISO/TS 15066:2016 Robots and robotic devices – Collaborative robots. Standard, International Organization for Standardization, Geneva, CH, Feb. 2016.

[71] S. Javdani, H. Admoni, S. Pellegrinelli, S. S. Srinivasa, and J. A. Bagnell. Shared autonomy via hindsight optimization for teleoperation and teaming. *The International Journal of Robotics Research*, 37(7):717–742, 2018.

[72] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011.

[73] M. Kallman and M. Mataric. Motion planning using dynamic roadmaps. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 5, pages 4399–4404, 2004.

[74] K. Kamali, I. A. Bonev, and C. Desrosiers. Real-time Motion Planning for Robotic Teleoperation Using Dynamic-goal Deep Reinforcement Learning. In *2020 17th Conference on Computer and Robot Vision (CRV)*, pages 182–189, 2020.

[75] A. Kanazawa, J. Kinugawa, and K. Kosuge. Adaptive motion planning for a collaborative robot based on prediction uncertainty to enhance human safety and work efficiency. *IEEE Transactions on Robotics*, 35(4):817–832, 2019.

[76] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[77] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[78] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, 5:90 – 98, 1985.

[79] J.-J. Kim and J.-J. Lee. Trajectory Optimization With Particle Swarm Optimization for Manipulator Motion Planning. *IEEE Transactions on Industrial Informatics*, 11(3):620–631, 2015.

[80] M.-C. Kim and J.-B. Song. Informed RRT* with improved converging rate by adopting wrapping procedure. *Intelligent Service Robotics*, 11(1):53–60, Jan 2018.

[81] K. Klasing, D. Wollherr, and M. Buss. Cell-based probabilistic roadmaps (CPRM) for efficient path planning in large environments. In *Proceedings of the 2007 International Conference on Advanced Robotics*, 2007.

[82] A. Knobloch, N. Vahrenkamp, M. Wächter, and T. Asfour. Distance-Aware Dynamically Weighted Roadmaps for Motion Planning in Unknown Environments. *IEEE Robotics and Automation Letters*, 3(3):2016–2023, 2018.

[83] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[84] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, 2005.

[85] S. Koenig, M. Likhachev, and D. Furcy. Lifelong Planning A*. *Artificial Intelligence*, 155(1):93–146, 2004.

[86] K. Kolur, S. Chintalapudi, B. Boots, and M. Mukadam. Online motion planning over multiple homotopy classes with gaussian process inference. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2358–2364, 2019.

[87] J. J. Kuffner and S. M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, volume 2, pages 995–1001, 2000.

[88] T. Kunz, U. Reiser, M. Stilman, and A. Verl. Real-time path planning for a robot arm in changing environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5906–5911, 2010.

[89] T. Kunz and M. Stilman. Time-Optimal Trajectory Generation for Path Following with Bounded Acceleration and Velocity. In *Robotics: Science and Systems VIII*. The MIT Press, 07 2013.

[90] R. Laha, W. Wu, R. Sun, N. Mansfeld, L. F. Figueredo, and S. Haddadin. S*: On safe and time efficient robot motion planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12758–12764, 2023.

[91] P. A. Lasota and J. A. Shah. Analyzing the effects of human-aware motion planning on close-proximity human–robot collaboration. *Human Factors*, 57(1):21–33, 2015.

[92] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, USA, 1991.

[93] S. LaValle. Rapidly-exploring random trees: a new tool for path planning. *The annual research report*, 1998.

[94] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, USA, 2006.

[95] P. Leven and S. Hutchinson. Toward real-time path planning in changing environments. In *Algorithmic and Computational Robotics:*

New Directions: The Fourth International Workshop on the Algorithmic Foundations of Robotics, pages 363–376, 2001.

[96] P. Leven and S. Hutchinson. A Framework for Real-time Path Planning in Changing Environments. *The International Journal of Robotics Research*, 21(12):999–1030, 2002.

[97] C. Li, F. Meng, H. Ma, J. Wang, and M. Q.-H. Meng. Relevant Region sampling strategy with adaptive heuristic for asymptotically optimal path planning. *Biomimetic Intelligence and Robotics*, 3(3):100113, 2023.

[98] S. Li and J. A. Shah. Safe and Efficient High Dimensional Motion Planning in Space-Time with Time Parameterized Prediction. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5012–5018, 2019.

[99] T.-Y. Li and Y.-C. Shie. An incremental learning approach to motion planning with roadmap management. In *Journal of Information Science and Engineering*, volume 23, pages 3411 – 3416, 2002.

[100] Z. Li, L. Ding, B. You, and H. Gao. A Local Dynamic Path Planning Approach for WMRs Based on Fuzzy Dual CHOMP. In *2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 491–496, 2019.

[101] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling*, page 262–271, 2005.

[102] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In *Proceedings of the 16th In-*

ternational Conference on Neural Information Processing Systems, page 767–774, 2003.

[103] J. Lim, S. Srinivasa, and P. Tsiotras. Lazy Lifelong Planning for Efficient Replanning in Graphs with Expensive Edge Evaluation. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8778–8783, 2022.

[104] H. Liu, X. Deng, H. Zha, and D. Ding. A Path Planner in Changing Environments by Using W-C Nodes Mapping Coupled with Lazy Edges Evaluation. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4078–4083, 2006.

[105] Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[106] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[107] R. Luna, I. A. Şucan, M. Moll, and L. E. Kavraki. Anytime solution optimization for sampling-based motion planning. In *2013 IEEE International Conference on Robotics and Automation*, pages 5068–5074, 2013.

[108] V. Magnanimo, S. Walther, L. Tecchia, C. Natale, and T. Guhl. Safeguarding a mobile manipulator using dynamic safety fields. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2972–2977, 2016.

[109] J. Mainprice, E. Akin Sisbot, L. Jaillet, J. Cortés, R. Alami, and T. Siméon. Planning human-aware motions using a sampling-based costmap planner. In *2011 IEEE International Conference on Robotics and Automation*, pages 5012–5017, 2011.

[110] J. A. Marvel and R. Norcross. Implementing speed and separation monitoring in collaborative robot workcells. *Robotics and Computer-Integrated Manufacturing*, 44:144–155, 2017.

[111] A. A. Maw, M. Tyan, and J.-W. Lee. iADA*: Improved Anytime Path Planning and Replanning Algorithm for Autonomous Vehicle. *Journal of Intelligent & Robotic Systems*, 100(3):1005–1013, Dec 2020.

[112] R. Meziane, M. J.-D. Otis, and H. Ezzaidi. Human-robot collaboration while sharing production activities in dynamic environment: SPADER system. *Robotics and Computer-Integrated Manufacturing*, 48:243–253, 2017.

[113] G. Michalos, P. Karagiannis, N. Dimitropoulos, D. Andronas, and S. Makris. *Human Robot Collaboration in Industrial Environments*, pages 17–39. Springer International Publishing, 2022.

[114] G. Nicola and S. Ghidoni. Deep Reinforcement Learning for Motion Planning in Human Robot cooperative Scenarios. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 01–07, 2021.

[115] C. Nielsen and L. Kavraki. A two level fuzzy PRM for manipulation planning. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1716–1721, 2000.

[116] N. J. Nilsson. *A mobile automaton: An application of artificial intelligence techniques*. Stanford research institute, 1969.

[117] C. Nissoux, T. Simeon, and J.-P. Laumond. Visibility based probabilistic roadmaps. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients*, volume 3, pages 1316–1321, 1999.

[118] O. Nocentini, L. Fiorini, G. Acerbi, A. Sorrentino, G. Mancioppi, and F. Cavallo. A Survey of Behavioral Models for Social Robots. *Robotics*, 8(3), 2019.

[119] O. S. Oguz, O. C. Sari, K. H. Dinh, and D. Wollherr. Progressive stochastic motion planning for human-robot interaction. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 1194–1201, 2017.

[120] U. Othman and E. Yang. Human - Robot Collaborations in Smart Manufacturing Environments: Review and Outlook. *Sensors*, 23(12), 2023.

[121] M. Otte and E. Frazzoli. RRTx: Real-time motion planning/replanning for environments with unpredictable obstacles. *The International Journal of Robotics Research*, 35(7):797–822, 2016.

[122] A. Palleschi, M. Hamad, S. Abdolshah, M. Garabini, S. Haddadin, and L. Pallottino. Fast and safe trajectory planning: Solving the cobot performance/safety trade-off in human-robot shared environments. *IEEE Robotics and Automation Letters*, 6(3):5445–5452, 2021.

[123] J. Pan and D. Manocha. GPU-based parallel collision detection for fast motion planning. *The International Journal of Robotics Research*, 31(2):187–200, 2012.

[124] C. Park, J. Pan, and D. Manocha. ITOMP: Incremental Trajectory Optimization for Real-Time Replanning in Dynamic Environments. *Proceedings of the International Conference on Automated Planning and Scheduling*, 2012.

[125] C. Park, J. Pan, and D. Manocha. Real-time optimization-based planning in dynamic environments using GPUs. In *2013 IEEE In-*

ternational Conference on Robotics and Automation, pages 4090–4097, 2013.

[126] M. G. Park, J. H. Jeon, and M. C. Lee. Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. In *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings*, volume 3, pages 1530–1535 vol.3, 2001.

[127] S. Petti and T. Fraichard. Safe motion planning in dynamic environments. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2210–2215, 2005.

[128] H. Pham and Q.-C. Pham. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, 34(3):645–659, 2018.

[129] A. Piazzi and A. Visioli. Global minimum-time trajectory planning of mechanical manipulators using interval analysis. *International Journal of Control*, 71(4):631–652, 1998.

[130] A. Piazzi and A. Visioli. Global minimum-jerk trajectory planning of robot manipulators. *IEEE Transactions on Industrial Electronics*, 47(1):140–149, 2000.

[131] E. Plaku, K. E. Bekris, and L. E. Kavraki. OOPS for Motion Planning: An Online, Open-source, Programming System. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3711–3716, 2007.

[132] I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3):193–204, 1970.

[133] I. Pohl. The Avoidance of (Relative) Catastrophe, Heuristic Competence, Genuine Dynamic Weighting and Computational Issues

in Heuristic Problem Solving. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, page 12–17, 1973.

[134] S. Proia, R. Carli, G. Cavone, and M. Dotoli. Control techniques for safe, ergonomic, and efficient human-robot collaboration in the digital industry: A survey. *IEEE Transactions on Automation Science and Engineering*, 19(3):1798–1819, 2022.

[135] J. Qi, H. Yang, and H. Sun. MOD-RRT*: A Sampling-Based Algorithm for Robot Path Planning in Dynamic Environment. *IEEE Transactions on Industrial Electronics*, 68(8):7244–7251, 2021.

[136] T. Qiao, D. Yang, W. Hao, J. Yan, and R. Wang. Trajectory planning of manipulator based on improved genetic algorithm. *Journal of Physics: Conference Series*, 1576(1):012035, jun 2020.

[137] C. Qixin, H. Yanwen, and Z. Jingliang. An Evolutionary Artificial Potential Field Algorithm for Dynamic Path Planning of Mobile Robot. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3331–3336, 2006.

[138] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[139] S. Quinlan. Real time modification of collision-free paths. *Stanford University*, 1994.

[140] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip. Motion Planning Networks: Bridging the Gap Between Learning-Based and Classical Motion Planners. *IEEE Transactions on Robotics*, 37(1):48–66, 2021.

[141] I. G. Ramirez-Alpizar, K. Harada, and E. Yoshida. Motion planning for dual-arm assembly of ring-shaped elastic objects. In *2014*

IEEE-RAS International Conference on Humanoid Robots, pages 594–600, 2014.

[142] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494, 2009.

[143] S. Robla-Gómez, V. M. Becerra, J. R. Llata, E. González-Sarabia, C. Torre-Ferrero, and J. Pérez-Oria. Working together: A review on safe human-robot collaboration in industrial environments. *IEEE Access*, 5:26754–26773, 2017.

[144] M. Safeea, P. Neto, and R. Bearee. On-line collision avoidance for collaborative robot manipulators by adjusting off-line generated paths: An industrial use case. *Robotics and Autonomous Systems*, 119:278–288, 2019.

[145] S. Sandrini and M. Beschi. An open-source ROS-Based human pose tracking system based on Mediapipe and Kalman Filter. https://github.com/SamueleSandrini/skeleton_trajectory.

[146] S. Sandrini, M. Faroni, and N. Pedrocchi. Learning action duration and synergy in task planning for human-robot collaboration. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–6, 2022.

[147] B. Sangiovanni, G. P. Incremona, M. Piastra, and A. Ferrara. Self-Configuring Robot Path Planning With Obstacle Avoidance via Deep Reinforcement Learning. *IEEE Control Systems Letters*, 5(2):397–402, 2021.

[148] B. Sangiovanni, A. Rendiniello, G. P. Incremona, A. Ferrara, and M. Piastra. Deep Reinforcement Learning for Collision Avoidance

of Robotic Manipulators. In *2018 European Control Conference (ECC)*, pages 2063–2068, 2018.

[149] H. Schumann-Olsen, M. Bakken, Ø. H. Holhjem, and P. Risholm. Parallel dynamic roadmaps for real-time motion planning in complex dynamic scenes. In *3rd Workshop on Robots in Clutter, IEEE*, 2014.

[150] Z. Shen, J. Wilson, R. Harvey, and S. Gupta. SMARRT: Self-Repairing Motion-Reactive Anytime RRT for Dynamic Environments. *ArXiv*, 2021.

[151] A. Short, Z. Pan, N. Larkin, and S. van Duin. Recent progress on sampling based dynamic motion planning algorithms. In *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1305–1311, 2016.

[152] E. A. Sisbot and R. Alami. A human-aware manipulation planner. *IEEE Transactions on Robotics*, 28(5):1045–1057, 2012.

[153] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, volume 4, pages 3310–3317, 1994.

[154] A. Stentz. The Focussed D* Algorithm for Real-Time Replanning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, page 1652–1659, 1995.

[155] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. http://ompl.kavrakilab.org.

[156] J. Sun, G. Liu, G. Tian, and J. Zhang. Smart Obstacle Avoidance Using a Danger Index for a Dynamic Environment. *Applied Sciences*, 9(8), 2019.

[157] W. Sun, S. Patil, and R. Alterovitz. High-frequency replanning under uncertainty using parallel sampling-based motion planning. *IEEE Transactions on Robotics*, 31(1):104–116, 2015.

[158] M. G. Tamizi, M. Yaghoubi, and H. Najjaran. A review of recent trend in motion planning of industrial robots. *International Journal of Intelligent Robotics and Applications*, 7(2):253–274, Jun 2023.

[159] S. Tarbouriech and W. Suleiman. Bi-objective Motion Planning Approach for Safe Motions: Application to a Collaborative Robot. *Journal of Intelligent & Robotic Systems*, 99(1):45–63, Jul 2020.

[160] C. Tonola and M. Beschi. OpenMORE: an Open-source MOtion REplanning library. https://github.com/JRL-CARI-CNR-UNIBS/OpenMORE.git.

[161] C. Tonola, M. Beschi, M. Faroni, and N. Pedrocchi. Openmore: an open-source tool for sampling-based path replanning in ros. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, 2023.

[162] C. Tonola, M. Faroni, S. Abdolshah, M. Hamad, S. Haddadin, N. Pedrocchi, and M. Beschi. Reactive and safety-aware path replanning for collaborative applications. Unpublished.

[163] C. Tonola, M. Faroni, M. Beschi, and N. Pedrocchi. Anytime informed multi-path replanning strategy for complex environments. *IEEE Access*, 11:4105–4116, 2023.

[164] C. Tonola, M. Faroni, N. Pedrocchi, and M. Beschi. Anytime informed path re-planning and optimization for human-robot collaboration. In *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, pages 997–1002, 2021.

[165] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 2, pages 1178–1183, 2003.

[166] J. van den Berg, D. Ferguson, and J. Kuffner. Anytime path planning and replanning in dynamic environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2366–2371, 2006.

[167] J. P. van den Berg. *Path planning in dynamic environments*. PhD thesis, Utrecht University, 2007.

[168] J. Vannoy and J. Xiao. Real-time adaptive and trajectory-optimized manipulator motion planning. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 497–502 vol.1, 2004.

[169] J. Vannoy and J. Xiao. Real-time adaptive motion planning (RAMP) of mobile manipulators in dynamic environments with unforeseen changes. *IEEE Transactions on Robotics*, 24(5):1199–1212, 2008.

[170] F. Vicentini. Terminology in safety of collaborative robotics. *Robotics and Computer-Integrated Manufacturing*, 63:101921, 2020.

[171] W. Wang, M. Zhu, X. Wang, S. He, J. He, and Z. Xu. An improved artificial potential field method of trajectory planning and obstacle avoidance for redundant manipulators. *International Journal of Advanced Robotic Systems*, 15(5), 2018.

[172] Y. Wang, L. Wei, K. Du, G. Liu, Q. Yang, Y. Wei, and Q. Fang. An online collision-free trajectory generation algorithm for human–robot collaboration. *Robotics and Computer-Integrated Manufacturing*, 80:102475, 2023.

[173] T. Weerakoon, K. Ishii, and A. A. F. Nassiraei. An Artificial Potential Field Based Mobile Robot Navigation Method To Prevent From Deadlock. *Journal of Artificial Intelligence and Soft Computing Research*, 5(3):189–203, 2015.

[174] C. Weidemann, N. Mandischer, F. van Kerkom, B. Corves, M. Hüsing, T. Kraus, and C. Garus. Literature Review on Recent Trends and Perspectives of Collaborative Robotics in Work 4.0. *Robotics*, 12(3), 2023.

[175] Y.-H. Wu, Z.-C. Yu, C.-Y. Li, M.-J. He, B. Hua, and Z.-M. Chen. Reinforcement learning in dual-arm trajectory planning for a free-floating space robot. *Aerospace Science and Technology*, 98:105657, 2020.

[176] E. K. Xidias. Time-optimal trajectory planning for hyper-redundant manipulators in 3D workspaces. *Robotics and Computer-Integrated Manufacturing*, 50:286–298, 2018.

[177] X. Xu, Y. Hu, J. Zhai, L. Li, and P. Guo. A novel non-collision trajectory planning algorithm based on velocity potential field for robotic manipulator. *International Journal of Advanced Robotic Systems*, 15(4), 2018.

[178] Y. Xu, A. Fern, and S. W. Yoon. Discriminative learning of beam-search heuristics for planning. In *International Joint Conference on Artificial Intelligence*, pages 2041–2046, 2007.

[179] Z. Xu, X. Zhou, H. Wu, X. Li, and S. Li. Motion Planning of Manipulators for Simultaneous Obstacle Avoidance and Target Tracking: An RNN Approach With Guaranteed Performance. *IEEE Transactions on Industrial Electronics*, 69(4):3887–3897, 2022.

[180] Y. Yang and O. Brock. Elastic roadmaps-motion generation for autonomous mobile manipulation. *Autonomous Robots*, 28(1):113–130, Jan 2010.

[181] K.-C. Ying, P. Pourhejazy, C.-Y. Cheng, and Z.-Y. Cai. Deep learning-based optimization for motion planning of dual-arm assembly robots. *Computers & Industrial Engineering*, 160:107603, 2021.

[182] H. You, G. Chen, Q. Jia, and Z. Huang. Path planning for robot in multi-dimensional environment based on dynamic prm blended potential field. In *2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*, volume 5, pages 1157–1162, 2021.

[183] C. Yuan, G. Liu, W. Zhang, and X. Pan. An efficient RRT cache method in dynamic environments for path planning. *Robotics and Autonomous Systems*, 131:103595, 2020.

[184] C. Yuan, G. Liu, W. Zhang, and X. Pan. An efficient RRT cache method in dynamic environments for path planning. *Robotics and Autonomous Systems*, 131:103595, 2020.

[185] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias. Safety in human-robot collaborative manufacturing environments: Metrics and control. *IEEE Transactions on Automation Science and Engineering*, 13(2):882–893, 2016.

[186] C. Zhang, J. Huh, and D. D. Lee. Learning Implicit Sampling Distributions for Motion Planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3654–3661, 2018.

[187] Z. Zhang, S. Chen, X. Zhu, and Z. Yan. Two Hybrid End-Effector Posture-Maintaining and Obstacle-Limits Avoidance Schemes for

Redundant Robot Manipulators. *IEEE Transactions on Industrial Informatics*, 16(2):754–763, 2020.

[188] 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 Journal for Science and Engineering*, 2021.

[189] M. Zhao and X. Lv. Improved Manipulator Obstacle Avoidance Path Planning Based on Potential Field Method. *Journal of Robotics*, 2020:1701943, Jan 2020.

[190] X. Zhao and J. Pan. Considering human behavior in motion planning for smooth human-robot collaboration in close proximity. In *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 985–990, 2018.

[191] R. Zhou and E. A. Hansen. Multiple Sequence Alignment Using Anytime A*. In *Eighteenth National Conference on Artificial Intelligence*, page 975–976, USA, 2002. American Association for Artificial Intelligence.

[192] Q. Zhu, Y. Yan, and Z. Xing. Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing. In *Sixth International Conference on Intelligent Systems Design and Applications*, volume 2, pages 622–627, 2006.

[193] M. Zucker, J. Kuffner, and J. A. Bagnell. Adaptive workspace biasing for sampling-based planners. In *2008 IEEE International Conference on Robotics and Automation*, pages 3757–3762, 2008.

[194] M. Zucker, J. Kuffner, and M. Branicky. Multipartite RRTs for rapid replanning in dynamic environments. *Proceedings of the IEEE In-*

ternational Conference on Robotics and Automation, pages 1603–1609, 2007.