T-PRM: Temporal Probabilistic Roadmap for Path Planning in Dynamic Environments

Matthias Hüppi, Luca Bartolomei, Ruben Mascaro and Margarita Chli
Vision For Robotics Lab, ETH Zürich, Switzerland

***Abstract*— Sampling-based motion planners are widely used in robotics due to their simplicity, flexibility and computational efficiency. However, in their most basic form, these algorithms operate under the assumption of static scenes and lack the ability to avoid collisions with dynamic (i.e. moving) obstacles. This raises safety concerns, limiting the range of possible applications of mobile robots in the real world. Motivated by these challenges, in this work we present *Temporal-PRM*, a novel sampling-based path-planning algorithm that performs obstacle avoidance in dynamic environments. The proposed approach extends the original Probabilistic Roadmap (PRM) with the notion of time, generating an augmented graph-like structure that can be efficiently queried using a time-aware variant of the A* search algorithm, also introduced in this paper. Our design maintains all the properties of PRM, such as the ability to perform multiple queries and to find smooth paths, while circumventing its downside by enabling collision avoidance in highly dynamic scenes with a minor increase in the computational cost. Through a series of challenging experiments in highly cluttered and dynamic environments, we demonstrate that the proposed path planner outperforms other state-of-the-art sampling-based solvers. Moreover, we show that our algorithm can run onboard a flying robot, performing obstacle avoidance in real time.**

I. Introduction

Collision-free motion planning is a challenging, yet essential requirement for any robotic system that is conceived to operate autonomously. With robots becoming ubiquitous outside the traditionally controlled industrial environments and the emergence of new fields, such as unmanned driving or intelligent logistics, research in path-planning strategies dealing with more complex, potentially dynamic scenarios has gained increasing interest in recent years. Especially, as mobile robots start to share their workspace with humans, the development of algorithms able to plan safe trajectories efficiently, while accounting for multiple moving obstacles is, in fact, of utmost importance.

The path-planning problem has historically been addressed in the literature from various perspectives, ranging from optimization- to sampling-based approaches. In contrast to the former, which require precise modelling of the robotic platform and its workspace to maintain the optimality guarantees, the latter can work in the robot's configuration space without an explicit representation of the environmental constraints. This characteristic makes sampling-based methods very efficient, flexible and particularly capable of dealing with high dimensional settings, even in environments containing a large number of obstacles. Among this class of path planners, the two most influential algorithms to date are the Probabilistic Roadmap (PRM) approach [1] and the Rapidly-exploring Random Tree (RRT) method [2]. While RRT-based approaches employ incremental sampling and search schemes, PRM and its variants rely on substantial pre-computation on the given environment, allowing for extremely fast, multiple planning queries. The need for information to avoid collisions during the pre-processing stage, however, typically makes PRM-based algorithms less suitable for deployment in dynamic scenarios. Even though a few works have aimed at tackling this issue in the past, these have, so far, resulted in methods that only support single-query execution modes [3] or require additional sampling and collision checking during the query phase, slowing down the path search process [4].

Fig. 1. Experiment with two UAVs flying in the same workspace. T-PRM successfully navigates one UAV to its destination, avoiding collision with the second, moving UAV, modeled as a spherical obstacle and highlighted with a red ellipse in the image. Our algorithm performs re-planning online, continuously adjusting the first UAV's trajectory to the newly estimated velocity of the obstacle.

In this paper, we present a novel and efficient multi-query approach to path planning that takes both static and dynamic obstacles into consideration. Coined as *Temporal PRM*, or *T-PRM* in short, our method extends the standard PRM formulation by incorporating the time intervals during which each node of the roadmap is not in collision with a dynamic obstacle. In addition, we introduce a variant of the A* search algorithm that considers these nodes' *time availability* and respects the time monotonicity constraint along the solution path. This allows for fast and efficient queries and removes the need for performing further sampling or rebuilding the graph, even in the presence of moving obstacles. The proposed approach is thoroughly evaluated in a simulated environment containing multiple static and

This work was supported by the Swiss National Science Foundation (SNSF, Agreement no. PP00P2183720), NCCR Digital Fabrication and the HILTI group.

Video – `https://youtu.be/Eh6brn_dVlU`
Code – `https://github.com/VIS4ROB-lab/t_prm`

dynamic obstacles, showing its ability to produce safer and more efficient paths than other state-of-the-art sampling-based planners, especially when dealing with highly dynamic settings. Furthermore, we demonstrate that our algorithm can be successfully deployed on an Unmanned Aerial Vehicle (UAV) to perform real-time obstacle avoidance in a series of challenging experiments.

In short, the main contributions of this work are:

- T-PRM, a novel and efficient algorithm for real-time obstacle avoidance in static and dynamic environments,
- an improved variant of A*, able to solve queries in graphs augmented with the time dimension,
- an extensive evaluation of the proposed method, both in simulation and in real-world experiments, and
- the source code of the proposed algorithm.

II. Related Work

Sampling-based motion planning is one of the most studied research topic in robotics [5], [6]. This class of motion planners is generally preferred to more complex algorithms, such as optimization-based solutions [7], [8], which require precise modelling of the problem, as well as the computation of complex constrained cost functions. Instead, sampling-based approaches are computationally more efficient. Although many different planners have been proposed over the years, the most important milestones in the field are Probabilistic Roadmaps (PRMs) [1] and Rapidly exploring Random Trees (RRTs) [2]. Both methods can solve high-dimensional planning problems and are able to generate collision-free paths from a start to a goal configuration, but they tackle the planning problem differently. RRTs iteratively build a tree rooted at the start configuration, growing it towards the goal by connecting randomly sampled states. On the other hand, PRMs divide the process into two separate phases. First, a roadmap is generated by wiring randomly sampled configurations to a graph, which is then queried to find a path from start to goal. The success of the aforementioned algorithms in dealing with complex environments and configuration spaces has lead to powerful variants based on both methods appearing in the literature. Within the paradigm of RRT-based approaches, RRT* [9], an asymptotically optimal version of RRT, together with its variants [10], [11] are widely used planners. Recently, extensions with deep reinforcement learning have been proposed, such as [12], where Q-Learning is used to guide the sampling process when expanding the tree. However, these approaches require expensive training and lack generalizability, as they target specific environment layouts. Whereas all these RRT-based algorithms assume a static environment, limiting their applicability in dynamic scenarios, there exist more advanced approaches specifically designed to plan in dynamic environments. RRTx [13] refines and repairs the same tree over the entire duration of navigation, while RRT*-FND [14] improves RRT*, making it usable for fast online re-planning when a moving obstacle blocks the path. Instead, Fisher *et al.* [15] use a combination of reachability maps and RRT-connect, while Ma'Arif *et al.* [16] use artificial potential fields to be reactive to dynamic obstacles. As in the case of RRTs, numerous methods following the PRM paradigm have also been proposed, targeting specific flaws of the original algorithm. In particular, PRM's most relevant limitations are the inability to find paths through narrow gaps, as well as to plan safe paths in dynamic environments. While there exist valid approaches that target the so-called *narrow gap problem* [17], performing dynamic obstacle avoidance with PRMs still remains an open research question. Lazy Toggle PRM [3] performs fewer collision checks, but it loses the multi-query property of PRMs. You *et al.* [4] propose to modify the positions of the nodes in the roadmap using potential fields to accommodate the changes in the environment caused by dynamic obstacles. However, this implies the necessity of additional collision checks when querying the graph, slowing down the algorithm. Instead, [18] proposes a method that can plan safe trajectories avoiding moving obstacles in human-robot collaborative environments; nonetheless, it loses all multi-query properties. To avoid these shortcomings, [19] use supervised learning to identify safe areas in dynamic scenes, reducing the search space and increasing the roadmap's coverage. However, this approach requires training data specific for the environments where the planner is deployed. Instead, similarly to our method, the planner by Phillips *et al.* [20] is able to navigate robots in dynamic scenes by introducing the notion of safe time intervals. However, their approach cannot re-plan in real time and does not scale well to large environments.

Motivated by these challenges, in this work we propose a PRM-based algorithm, dubbed T-PRM, able to generate safe paths avoiding collisions with both static and dynamic obstacles. In a nutshell, we propose to extend the idea of roadmaps with the notion of time, in order to be reactive in changing environments. Even though this implies substantial changes to the original algorithm, the proposed strategy maintains all the properties of PRMs, while adding dynamic obstacle avoidance capabilities. Our approach generates smoother and safer paths than other state-of-the-art planners, and it can be used on embedded platforms to perform path planning in real-time.

III. Methodology

Our objective is to generate a safe path from a starting location to a given destination, avoiding collisions with both static and moving obstacles. We tackle this problem using a novel adaptation of Probabilistic Roadmaps, firstly introduced by Kavraki *et al.* [1] for static environments, which we modify in order to account for the presence of dynamic obstacles in the workspace.

The core novelty lays in the extension of the original PRM algorithm to consider the time dimension when building and querying roadmaps, allowing us to perform dynamic obstacle avoidance. Our version, Temporal PRM or T-PRM, is developed for holonomic robots moving in an n-dimensional space $\mathbb{R}^n$, where both static and moving objects are encountered. For simplicity, we only treat first-order dynamics of the robots, which are modeled as points with no

physical extension. To ensure safety, we inflate obstacles by a safety margin, which is representative of the robot's dimensions. Moreover, in general, we assume that the obstacles' positions and velocities are known and time-invariant, i.e. constant, when building and querying roadmaps. However, we demonstrate that our approach, thanks to its efficiency, can also handle scenarios where obstacles change velocities and direction of travel unexpectedly.

In the following, we recap the main concepts of PRMs, and detail the proposed approach.

A. Preliminaries: Probabilistic Roadmaps

A *roadmap* is a data structure in the form of an undirected graph $G = (V, E)$ that consists of a set of nodes V connected by edges E. The graph G is firstly built in the free configuration space of the robot, where each node $v \in V$ represents a robot's configuration and edges $e = (v, w) \in E$ with $v, w \in V$ are feasible paths connecting different configurations. The PRM algorithm initially builds G (*learning phase*), and then queries it (*query phase*) in order to find the shortest and collision-free path between the start and goal configurations. During the learning phase, nodes are sampled randomly in the robot's free configuration space, i.e. not inside static obstacles. When a new node v is sampled, it is connected to the graph G if it is possible to generate a local path from v to the neighboring nodes already in G. A *local planner* is used to find a feasible path between node pairs and, when the search is successful, the resulting edge is added to E. This phase continues until the given time budget is exhausted or when the maximum number of samples is reached. In the query phase, the graph G is used to find paths between configurations. Since the original formulation of PRMs considers only static scenes, the graph is built only once. Therefore, the same underlying structure can be reused for multiple queries without additional sampling.

B. Temporal Probabilistic Roadmaps

Since PRMs assume environments to be static, the algorithm cannot be utilized to perform obstacle avoidance in dynamic scenes. In our formulation, we propose to incorporate the notion of time in the graph G, while keeping the learning and query phases separate, similarly to PRMs, and to differentiate static from moving obstacles.

1) Learning Phase: In order to build the graph G, samples are generated uniformly in the space. A candidate sample is considered valid if it does not lay in a static obstacle. While many sampling strategies are proposed in the literature [21], [22], these methods place a lower number of samples in wide open spaces. However, this can potentially be problematic in dynamic environments. As shown in Fig. 2, if a moving obstacle blocks a node without close-by neighbors, the majority of the map becomes restricted. By sampling uniformly the space at random, this problem can be circumvented at the expense of higher number of samples. Despite its simplicity, this sampling strategy is proven to work effectively in practice. Assuming the robot is holonomic, edges are then created connecting node pairs by means of a simple local planner that creates straight-line paths. If the connection does not exceed a maximum length and is feasible, i.e. not in collision with static obstacles, the newly built edge is added to E.

Algorithm 1: Learning Phase

Input: Obstacles Information, Max Number of Nodes.
Output: $G = (V, E)$ with $TA^v \ \forall v \in V$.

```
while not all nodes sampled do
    Sample v in free space (outside static obstacles).
    Compute TA^v for node v.
    Add node v to V.
    Generate edge e from v to its neighbors.
    if e is valid then
        Add e to E.
Return G = (V, E).
```

In order to accommodate also for the presence of moving objects, we introduce the concept of *time availability* of the nodes, TA in short. This encodes the time intervals, during which a given node is free (or *available*) and not in collision with a dynamic obstacle. Note that we do not assign TA to edges as a design choice. Since they have a length lower than the obstacles' dimensions, we argue that checking nodes is sufficient to find a safe path. However, our approach could be easily extended to check for collisions along edges as well. Formally, TA^v of node $v \in V$ is the set of continuous closed time intervals, during which node v is available, considering the set of N moving obstacles $\mathcal{O} := \{o_k\}_{k=1}^{N}$ in the scene. This can be computed as

$$TA^v = \bigcap_{o \in \mathcal{O}} TA_o^v, \tag{1}$$

where TA_o^v is the time availability for node v given obstacle o, i.e. when v is not blocked by o. The intersection operator in (1) implies that a node is available at a given time when no dynamic obstacle obstructs it. Since we assume obstacles have a known constant velocity, TA^v for all nodes $v \in V$ are directly computed when the graph G is constructed. This design choice is beneficial in case of multiple queries of G, as collisions against dynamic obstacles are checked only during the building phase. Computing the time availability of nodes when querying the graph makes the process slow, as the information about obstacles needs to be propagated through the structure multiple times. This would contradict the nature of PRMs, as they are specifically built to answer multiple queries efficiently. Notice that, since G is an undirected graph, time monotonicity is not directly encoded in the graph, but it is enforced during the query phase. The learning phase is summarized in Alg. 1.

2) Query Phase: During the query phase (Alg. 2), A* is used to find the shortest paths in the graph G between pairs of start and goal nodes, indicated with s and g, respectively. Every edge $e \in E$ is assigned a cost equal to its length, as well as an approximate total time to traverse it, which is computed assuming a holonomic robot with

Algorithm 2: Query Phase

Input: Graph G, start node s, goal node g
Output: Path from $s \to g$ with timings

Find closest node $\tilde{s}$ and $\tilde{g}$ in G for s and g, respectively.
Find path $\tilde{s} \to \tilde{g}$ using Alg. 3.
Concatenate the paths $s \to \tilde{s}$, $\tilde{s} \to \tilde{n}$, $\tilde{n} \to n$.
Return complete path $s \to g$.

Algorithm 3: A* variant. The Open Queue is sorted using the f-Cost, given by the Cost-to-go plus the heuristic score per node. The term *active* indicates that a node is not in collision at a given time.

Input: Graph G with $TA^v \; \forall v \in V$, start node s, goal node g, heuristic function h.
Output: Shortest path $s \to g$ with timings.

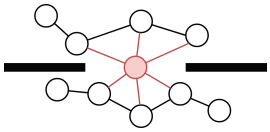
// Initialization of containers to empty lists
Open Queue, Closed List, Predecessors, Arrival Time, f-Cost, Cost-to-go $\leftarrow [\,]$
Push s to Open Queue
Arrival time[s] $\leftarrow 0$
Cost-to-go[s] $\leftarrow 0$
f-Cost[s] $\leftarrow h(s)$
// Check TA^s:
if s *is not active at* 0 **then**
    return $\{\}$ // Return empty path
while *Open Queue is not empty* **do**
    $v \leftarrow$ pop from Open Queue
    $t_v \leftarrow$ Arrival Time[v]
    if $v = g$ **then**
        break
    Add v to Closed List
    for *each outgoing edge* e *from* v **do**
        $w \leftarrow$ end node of e
        // Check TA^w:
        if w *is active at* t_v + *duration*(e) **then**
            Tentative Cost-to-go $\leftarrow$ Cost-to-go[v] + cost(e)
            if w *in Open Queue and Tentative Cost-to-go* $>$ *Cost-to-go*[w] **then**
                continue
            if w *is not in Closed List* **then**
                Predecessors[w] $\leftarrow v$
                Arrival Time[w] $\leftarrow t_v$ + duration(e)
                Cost-to-go[w] $\leftarrow$ Tentative Cost-to-go
                f-Cost[w] $\leftarrow$ Tentative Cost-to-go + h(w)
                if w *not in Open Queue* **then**
                    Push w to Open Queue
Back-trace path $s \to g$ using Predecessors and Arrival Times

constant velocity. However, since time availability of nodes is represented with a set of continuous time intervals, the standard formulation of A* cannot be used because it does not keep track of the timings along the path. Therefore, we propose a novel version of the A* algorithm, reported in Alg. 3, that works with continuous time intervals and that respects the constraint of time monotonicity along the solution path. The standard version of A* can be utilized if time, instead of continuous, is discretized, e.g. into T steps. In this case, a series of new graphs $\{G_t\}_{t=0}^{T}$ can be constructed from G for each time step t by replacing each undirected edge with two directed ones. However, while a solution for every step t can be found, the graph size is increased and the overall performance of the algorithm is negatively affected. Moreover, this effect is exacerbated with higher time resolutions. In our variant of A*, we start at node s at time $t = 0$. We then march along the undirected edges, and we track for each node the arrival time, as well as its predecessor along the path. This bookkeeping allows to respect the time monotonicity constraint. By using the arrival times and time availability TA^v for each node v, we avoid reaching a node when it is blocked by a dynamic obstacle.

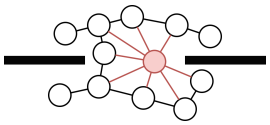
We give an example of the process in Fig. 3. As a heuristic cost for A*, we use the Euclidean distance to the goal node,

$$h(v) = \|v - g\|_2 \,. \tag{2}$$

Another benefit of our A* variant is that the output path already incorporates timings. This implies that no path re-parametrization has to be performed after its computation. Furthermore, since T-PRM inherits from PRM, it maintains the property of *probabilistic completeness*. In fact, the only consequence of the incorporation of time into the graph is a more expensive search of a solution path in the roadmap.



(a) Random sampling of nodes biased towards free space. Fewer nodes are placed in-between the walls. The invalid node and edges (in red) split the graph into two separate parts.

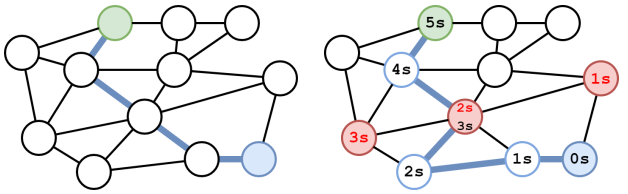


(b) Uniform random sampling strategy. Despite that the central node is blocked, the graph remains connected.

Fig. 2. Benefit of our sampling strategy. In (a), the red node is currently blocked by a dynamic obstacle and the corresponding edges (in red) are invalid, splitting the graph in two. In dynamic environments, a uniform random sampling strategy (b) is beneficial, as it allows to have redundant connections in case one node is blocked.

C. Multiple Queries and Re-planning

The original formulation of PRM can solve multiple queries in static environments, as the graph built in the learning phase can be reused without additional computations. Thanks to the assumption of known obstacles' velocities, T-PRM maintains this property also in dynamic scenes, as the movements of obstacles are already incorporated in the roadmap. In case this assumption is not respected, our algorithm can be adapted to perform re-planning at fixed rate by recomputing the time availability of nodes. This allows



(a) Case without dynamic obstacles.

(b) Case of a dynamic obstacle blocking some nodes (red) at different times.

Fig. 3. Examples of paths found by T-PRM highlighted in blue from start (blue) to goal (green) in static (a) and dynamic (b) scenes. When an obstacles moves in the workspace as in (b), T-PRM generates a timed path, which avoids collisions. In (b) each node is labeled with the robot's arrival time (black), assuming that all the edges take 1 s to traverse.

to keep the same underlying graph structure, at the expenses of minor computation overheads.

D. Complexity Analysis

Here we analyse the complexity of the T-PRM algorithm using the big-O notation by checking the complexities of the learning and query phases separately.

In the learning step, we compute the time availability of each node and create edges connecting node pairs. Assuming the sets of sampled nodes V and of moving obstacles $\mathcal{O}$ are given, at most $|\mathcal{O}| + 1$ time intervals have to be computed. This happens when every obstacle covers the node at a different interval, and implies that the complexity of computing the time availability of each node $v \in V$ is $O(|V| \cdot |\mathcal{O}|)$. In the second stage, we compute edges in $O(|V|^2)$ using a brute-force approach by checking all possible node pairs. Overall, the complexity of the learning phase is $O(|V|^2 + |V| \cdot |\mathcal{O}|))$. However, from a practical perspective, the number of moving obstacles $|\mathcal{O}|$ is generally magnitudes smaller than the number of nodes, that is $|\mathcal{O}| \ll |V|$. This implies that the complexity when building the roadmap G is dominated by the wiring process.

The complexity of the query phase coincides with the complexity of our A* variant. While the standard formulation of A* is in the order of $O(|V| \log |V| + |E|)$ [9], our version has worse complexity, since it needs to check the time availability of nodes. In the worst case scenario, where a node is blocked by all moving obstacle at different times, this leads to $|\mathcal{O}|+1$ time intervals. These intervals, if sorted, can be searched in $O(\log |\mathcal{O}|)$, using e.g. binary search. This implies that the query phase has a total complexity of $O(\log |\mathcal{O}| \cdot (|V| \log |V| + |E|))$.

In case of re-planning, recomputing the time availability of all the node has complexity $O(|V| \cdot |\mathcal{O}|)$, as described above.

IV. Experiments

We conduct a series of challenging experiments both in simulation and in the real world, and we compare our approach against our implementation of RRT*-FND [14] and the RRT* and PRM solvers from the Open Motion Planning Library (OMPL) [23]. We compare the computation times of the different algorithms, as well as the lengths of the generated paths in environments cluttered with a high number of simulated static and moving obstacles. The experiments demonstrate that T-PRM can find safer and more efficient paths than the competitors, even in highly dynamic environments, as shown in the accompanying video. Finally, by conducting a series of real-world experiments with an Unmanned Aerial Vehicle (UAV), we demonstrate that T-PRM can be deployed in reality to perform obstacle avoidance in real time.

TABLE I

Performances of T-PRM and competitors averaged over 100 runs with varying number of static obstacles in 2D. The best results are highlighted in bold.

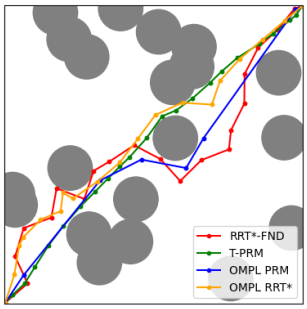
Planner	# Obstacles	Path Length [m]	Computation Time [ms]
T-PRM	0	**14.69 ± 0.16**	4.99 ± 1.22
OMPL PRM	0	15.34 ± 0.74	2.78 ± 1.49
OMPL RRT*	0	16.60 ± 0.85	**0.99 ± 0.69**
RRT*-FND	0	17.76 ± 1.16	4.32 ± 0.06
T-PRM	5	**14.68 ± 0.18**	4.19 ± 0.91
OMPL PRM	5	15.27 ± 0.78	2.76 ± 1.31
OMPL RRT*	5	16.44 ± 0.79	**1.01 ± 0.61**
RRT*-FND	5	17.82 ± 1.28	4.28 ± 0.10
T-PRM	10	**14.96 ± 0.19**	5.23 ± 1.01
OMPL PRM	10	15.32 ± 0.89	2.77 ± 1.19
OMPL RRT*	10	16.54 ± 0.79	**0.96 ± 0.52**
RRT*-FND	10	18.56 ± 1.30	4.09 ± 0.14
T-PRM	15	**15.04 ± 0.28**	5.66 ± 1.03
OMPL PRM	15	15.22 ± 0.61	2.84 ± 1.91
OMPL RRT*	15	16.57 ± 0.86	**0.97 ± 0.63**
RRT*-FND	15	17.97 ± 1.26	3.79 ± 0.21
T-PRM	20	15.30 ± 0.58	6.27 ± 0.68
OMPL PRM	20	**15.29 ± 0.69**	2.96 ± 1.72
OMPL RRT*	20	16.74 ± 0.96	**0.94 ± 0.56**
RRT*-FND	20	18.60 ± 1.30	3.69 ± 0.24

A. Experiments in Simulation

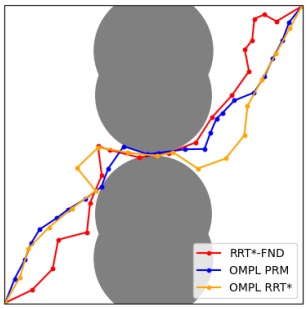
We task the planners to find a path from a starting location to a desired destination, and we conduct the experiments both in 2D and in 3D in environments filled with static and moving obstacles. The test areas have sizes $10\,\mathrm{m} \times 10\,\mathrm{m}$ and $10\,\mathrm{m} \times 10\,\mathrm{m} \times 10\,\mathrm{m}$, respectively. The final path has to connect one corner of the testing area to the diagonally opposite one (as shown in Fig. 4), while avoiding collisions. The algorithms are implemented in C++, and experiments and benchmarks are run using an Intel Core i7-4790 with 8 cores and 16GB of RAM.

1) Static Environments: In Tables I and II we report the results of the experiments in 2D and 3D, respectively, averaged over 100 runs, with different numbers of static obstacles randomly placed in the scene. In the learning phase of T-PRM, we set the maximum number of nodes to 800 in 2D and 1300 in 3D. The computation times of the different algorithms correspond to the time required to find a valid collision-free path. To have a fair comparison of the computation times of the different algorithms, we stop planning when the first valid solution is found.

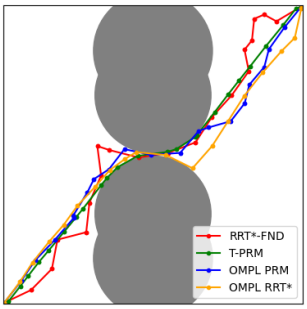
While the overall runtimes of the different planners are not affected by the number of obstacles, it is noticeable how T-PRM and PRM generate overall smoother and shorter paths compared to RRT*-FND and RRT*. Fig. 4(a) shows a qualitative comparison of the trajectories found by the different solvers in the 2D case with 20 obstacles. However,



(a) Planning with 20 static obstacles randomly placed in the space.



(b) Narrow gap with 1000 T-PRM nodes. T-PRM is not able to find a path.



(c) Same narrow gap as in (b) with 5000 T-PRM nodes. T-PRM is able to find a path.

Fig. 4. Paths generated by the different algorithms in a 2D square from bottom left to top right, where the intermediate dots represent the sampled nodes of the underlying data structure. (b) and (c) show the performance of T-PRM with a different number of samples in the presence of a narrow gap.

TABLE II

PERFORMANCES OF T-PRM AND COMPETITORS AVERAGED OVER 100 RUNS WITH VARYING NUMBER OF STATIC OBSTACLES IN 3D. THE BEST RESULTS ARE HIGHLIGHTED IN BOLD.

Planner	# Obstacles	Path Length [m]	Computation Time [ms]
T-PRM	0	**18.58 ± 0.32**	12.92 ± 6.06
OMPL PRM	0	19.94 ± 1.30	6.08 ± 4.72
OMPL RRT*	0	22.13 ± 1.80	**1.68 ± 0.93**
RRT*-FND	0	24.83 ± 2.37	30.38 ± 1.35
T-PRM	5	**18.55 ± 0.28**	12.71 ± 4.97
OMPL PRM	5	20.19 ± 1.63	6.07 ± 4.03
OMPL RRT*	5	22.04 ± 1.78	**1.55 ± 0.63**
RRT*-FND	5	24.96 ± 2.78	29.88 ± 1.62
T-PRM	10	**18.55 ± 0.30**	12.48 ± 5.46
OMPL PRM	10	20.62 ± 2.01	6.81 ± 4.57
OMPL RRT*	10	22.23 ± 1.84	**1.59 ± 0.66**
RRT*-FND	10	25.48 ± 2.71	29.39 ± 0.95
T-PRM	15	**18.69 ± 0.31**	15.73 ± 7.67
OMPL PRM	15	20.79 ± 1.60	6.36 ± 5.16
OMPL RRT*	15	22.18 ± 1.70	**1.76 ± 1.04**
RRT*-FND	15	24.78 ± 2.27	28.73 ± 1.00
T-PRM	20	**18.59 ± 0.29**	12.95 ± 5.15
OMPL PRM	20	20.66 ± 1.77	6.72 ± 4.59
OMPL RRT*	20	21.94 ± 1.86	**1.61 ± 0.80**
RRT*-FND	20	25.07 ± 2.19	29.22 ± 2.00

TABLE III

PERFORMANCE OF THE DIFFERENT ALGORITHMS AVERAGED OVER 100 RUNS IN DYNAMIC SCENES WITH UP TO 1000 OBSTACLES MOVING RANDOMLY IN 3D. RRT*-FND AND THE OMPL PLANNERS RE-PLAN IF THEIR PATHS ARE IN COLLISION WITH AN OBSTACLE. THE BEST RESULTS ARE HIGHLIGHTED IN BOLD.

Planner	# Obstacles	Path Length [m]	Computation Time [ms]	Success Rate [%]
T-PRM	50	**18.51 ± 0.26**	12.39 ± 4.42	99
OMPL PRM	50	21.19 ± 2.44	20.96 ± 16.25	**100**
OMPL RRT*	50	22.11 ± 1.65	**3.30 ± 1.69**	99
RRT*-FND	50	29.51 ± 4.86	87.78 ± 20.11	98
T-PRM	100	**18.55 ± 0.26**	13.14 ± 4.69	**99**
OMPL PRM	100	21.33 ± 2.21	28.59 ± 18.83	96
OMPL RRT*	100	22.40 ± 1.94	**4.08 ± 1.85**	**99**
RRT*-FND	100	29.90 ± 4.17	142.22 ± 33.49	98
T-PRM	500	**18.59 ± 0.33**	**17.01 ± 5.62**	**100**
OMPL PRM	500	25.69 ± 4.24	230.34 ± 123.14	84
OMPL RRT*	500	23.54 ± 3.12	19.35 ± 6.89	88
RRT*-FND	500	32.55 ± 6.03	654.52 ± 159.40	99
T-PRM	1000	**18.60 ± 0.31**	**20.93 ± 5.37**	**100**
OMPL PRM	1000	29.59 ± 6.37	757.95 ± 311.94	66
OMPL RRT*	1000	24.42 ± 3.16	51.32 ± 15.57	62
RRT*-FND	1000	36.77 ± 7.61	1589.18 ± 360.83	94

while T-PRM finds the shortest paths in most cases, RRT* has the lowest computation times. This is expected, as RRT* specifically targets path planning in static scenes, while the proposed T-PRM planner is designed to also perform dynamic obstacle avoidance.

Notice that we limit the maximum number of static obstacles to 20. This is motivated by the known inability of PRM to find a valid path when the only possible connection goes through a narrow gap [21]. While T-PRM shows promising performances in relatively open spaces, it suffers from the same limitation. This can be directly related to the uniform sampling strategy. By increasing the maximum number of allowed samples, this problem is overcome at the expense of higher computation times. In Fig. 4(b), we set the maximum number of nodes to 1000. In this case, the competitor planners manage to find a path, whereas T-PRM fails. This is to be expected, since the PRM implementation in OMPL is heavily optimized. If the number of nodes is increased to 5000, T-PRM is also able to find a solution (Fig. 4(c)), but in this case the computation time goes up to 200 ms. Notice that with a gap size 3 times larger T-PRM is able to find a connection using only 1000 nodes in 11 ms. In order to keep timings low, the *narrow gap* problem can be alleviated by employing multi-resolution uniform sampling strategies, e.g. placing nodes more densely around the gap.

2) Dynamic Environments: We run similar experiments in scenes with moving obstacles to demonstrate the benefits of T-PRM. Similarly to the previous set of experiments, we compare our approach against RRT*-FND and OMPL. Since OMPL is designed to handle only static environments, we check for collisions at 4 Hz and perform re-planning if the path is blocked. We run the experiments with an increasing number of dynamic obstacles, assuming the robot always moves with constant velocity. The obstacles are spawned in the navigation area randomly, and they are assigned a velocity sampled in the interval $[-0.2, 0.2]^3 \frac{\text{m}}{\text{s}}$. In Table III we summarize the results. Thanks to its ability to handle moving obstacles, T-PRM generates the shortest paths in all the experiments, with lower computation times than PRM and RRT*-FND. Moreover, when increasing the number of moving obstacles, our algorithm is faster than RRT* and reaches the highest success rates. Notice that we report the total time taken by the different solvers to generate valid

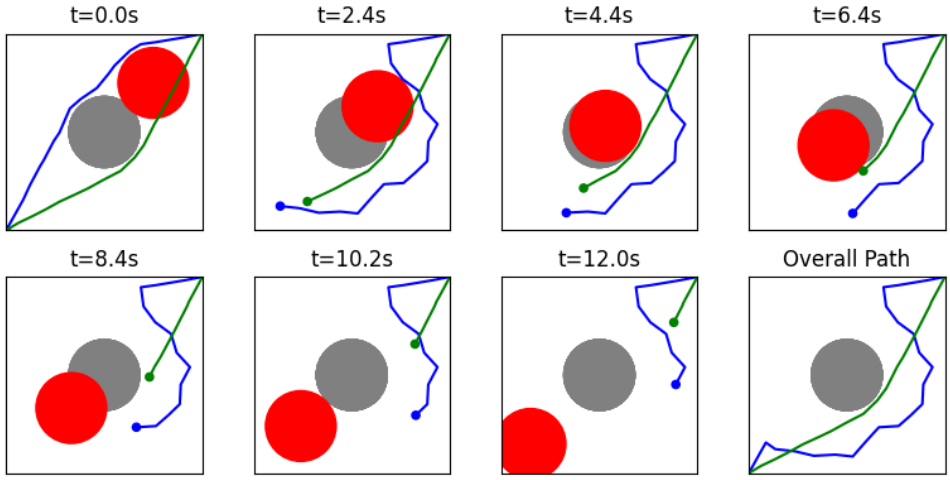


Fig. 5. Comparison of the paths from bottom left to top right shown at different time steps, as they get generated by T-PRM (green) and RRT*-FND (blue), with a static (gray) and a dynamic obstacle (red) moving in a $10\,\mathrm{m} \times 10\,\mathrm{m}$ scene. While the planners have a similar runtime ($40\,\mathrm{ms}$), T-PRM with 2000 nodes and a maximum edge length of $1\,\mathrm{m}$ generates a shorter path ($14.8\,\mathrm{m}$) than RRT*-FND ($21.2\,\mathrm{m}$), since the latter has to continuously adjust the trajectory to account for the updated obstacle position.

Fig. 6. Snapshot of the real-life experiment with a static obstacle, modeled as a pillar. A safety margin of 0.5m, shown as a semi-transparent rim in the inset, is added to the static obstacle. The starting position is shown as a blob, while the destination as a star.

paths over the whole run, i.e. summing the timings used to generate a path at every re-planning iteration. Fig. 5 shows the comparison between T-PRM and RRT*-FND as consecutive snapshots of a 2D experiment with a static (gray) and a moving obstacle (red), traversing the space and blocking the direct path from start the goal. While T-PRM finds a feasible connection in a single planning call, RRT*-FND needs to iteratively check the obstacle position and update its path. This is performed by reconnecting the tree and continuously regrowing it, resulting in higher traveled distances. Examples of similar experiments in complex scenes are shown in the accompanying video.

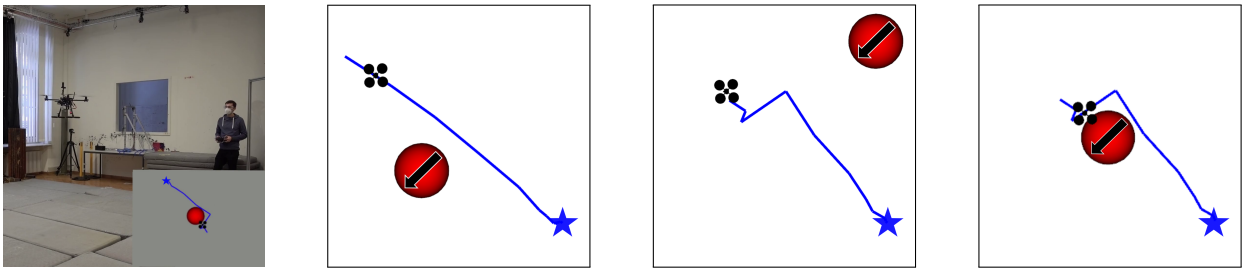
B. Real-World Experiments

We test our algorithm in a series of challenging real-world experiments, with both static and moving obstacles. The UAV used in the experiments is a *Holybro X500*, equipped with an Intel NUC with an Intel Core i5-1145G7 CPU. We use the *Pixhawk* autopilot[1] as a low-level controller, while state estimation is performed by fusing inertial data with VICON[2] measurements. All the computations are performed onboard. We task the drone to navigate autonomously from one side of the room to the opposite one multiple times. In a first experiment, the paths generated by T-PRM fly the drone at $0.5\,\frac{\mathrm{m}}{\mathrm{s}}$ around the static obstacle, inflated by a safety margin, placed at the center of the room (Fig. 6). By sampling $10\,000$ nodes, the algorithm takes $600\,\mathrm{ms}$ to build the initial graph, while every query of the roadmap consumes about $50\,\mathrm{ms}$.

In a second and more challenging experiment, we show that T-PRM can perform dynamic obstacle avoidance, by flying across the room multiple times dodging a virtual sphere moving with known constant velocity. To test the online computational capabilities of T-PRM, we manually trigger the re-planning to force the algorithm to generate a new path on command. With $10\,000$ nodes, T-PRM takes up about 40 to $50\,\mathrm{ms}$ to recompute the time availability of the nodes and query a new route, demonstrating that it can re-plan online at about $20\,\mathrm{Hz}$ on an embedded platform. In Fig. 7, we show successive snapshots of this experiment. In Fig. 7(b), a first path is generated and, even if the obstacle crosses it, no re-planning is required since the sphere and the drone are not on collision course. The obstacle is then reset to its starting location (Fig. 7(c)). Since in this case the spherical obstacle will collide with the robot, re-planning is triggered and a new path is found. By following the newly generated path, the UAV dodges the moving obstacle successfully (Fig. 7(d)).

In a final and most challenging experiment, we demonstrate the re-planning capabilities of T-PRM when a real moving obstacle with changing velocity has to be avoided, with a path and a velocity unknown to the planner. In Fig. 1, we show the path generated by T-PRM that successfully avoids the collision with a second drone flying in the workspace. In this case, the second robot was flown manually and it was modeled as a spherical obstacle with non-constant velocity, measured via the VICON system. Nevertheless, T-PRM was able to generate safe paths in real-time on the UAV's embedded computer, successfully

[1] https://pixhawk.org
[2] https://www.vicon.com



(a) A snapshot of the experiment.

(b) Initial path generated by T-PRM.

(c) The obstacle is re-spawned at its starting location.

(d) Dodging maneuver when the obstacle is in the robot's proximity.

Fig. 7. Example of a re-planning iteration (a). Initially, the obstacle crosses the computed path (b). The obstacle is then reset, re-planning is triggered (c), and the drone performs a dodging maneuver to avoid the obstacle (d). Notice that here the UAV's dimensions are exaggerated for visualization purposes. The destination is shown as a star.

performing computations online and navigating the robot to its destination.

V. Conclusion

In this work, we present T-PRM, a novel, sampling-based algorithm for path planning in dynamic environments. Our method extends the original Probabilistic Roadmap planner with the notion of time availability, i.e. encoding the continuous time intervals in which each node of the roadmap is not in collision with a dynamic obstacle. In addition, we introduce a variant of A* that simultaneously deals with the aforementioned time intervals and respects the time monotonicity constraint when searching for a solution path. T-PRM maintains all the original properties of PRM, while being able to perform dynamic obstacle avoidance. By means of a series of challenging experiments in dynamic scenes, we demonstrate that the proposed planner generates smoother paths within a lower computation time compared to other state-of-the-art algorithms. Moreover, we show that our planner can be used for path planning in real-time on a UAV, performing re-planning of trajectories online.

Future directions include investigating the extension of the proposed planner to non-holonomic robots considering the full dynamics and dropping the assumption of known obstacles' velocities by incorporating the uncertainty of the velocity estimates into the graph.

References

[1] 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*, 1996.

[2] S. M. LaValle, "Rapidly-Exploring Random Trees: a new Tool for Path Planning," *The annual research report*, 1998.

[3] J. Denny, K. Shi, and N. M. Amato, "Lazy Toggle PRM: A Single-query Approach to Motion Planning," in *2013 IEEE International Conference on Robotics and Automation*, 2013.

[4] 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)*, 2021.

[5] A. Short, Z. Pan, N. Larkin, and S. Duin, "Recent Progress on Sampling Based Dynamic Motion Planning Algorithms," *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, 2016.

[6] M. Radmanesh, M. Kumar, P. H. Guentert, and M. Sarim, "Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study," *Unmanned Systems*, 2018.

[7] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian Optimization for Motion Planning," *The International Journal of Robotics Research*, 2013.

[8] A. Byravan, B. Boots, S. S. Srinivasa, and D. Fox, "Space-Time Functional Gradient Optimization for Motion Planning," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[9] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *The International Journal of Robotics Research*, 2011.

[10] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[11] O. Arslan and P. Tsiotras, "Dynamic Programming Guided Exploration for Sampling-based Motion Planning Algorithms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[12] J. Huh and D. D. Lee, "Efficient Sampling With Q-Learning to Guide Rapidly Exploring Random Trees," *IEEE Robotics and Automation Letters*, 2018.

[13] M. Otte and E. Frazzoli, "RRTX: Asymptotically Optimal Single-query Sampling-based Motion Planning with Quick Replanning," *The International Journal of Robotics Research*, 2016.

[14] O. Adiyatov and H. A. Varol, "A Novel RRT*-based Algorithm for Motion Planning in Dynamic Environments," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2017.

[15] R. Fisher, B. Rosman, and V. Ivan, "Real-Time Motion Planning in Changing Environments Using Topology-Based Encoding of Past Knowledge," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[16] A. Ma'Arif, W. Rahmaniar, M. A. M. Vera, A. A. Nuryono, R. Majdoubi, and A. Çakan, "Artificial Potential Field Algorithm for Obstacle Avoidance in UAV Quadrotor for Dynamic Environment," in *2021 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, 2021.

[17] K. Cao, Q. Cheng, S. Gao, Y. Chen, and C. Chen, "Improved PRM for Path Planning in Narrow Passages," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2019.

[18] 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)*, 2019.

[19] F. F. Arias, B. Ichter, A. Faust, and N. M. Amato, "Avoidance critical probabilistic roadmaps for motion planning in dynamic environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[20] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[21] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners," in *2003 IEEE International Conference on Robotics and Automation*, 2003.

[22] R. Geraerts and M. H. Overmars, "Sampling and Node Adding in Probabilistic Roadmap Planners," *Robotics and Autonomous Systems*, 2006.

[23] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, 2012, https://ompl.kavrakilab.org.