

# Online Motion Planning Over Multiple Homotopy Classes with Gaussian Process Inference

Keshav Kolar\*, Sahit Chintalapudi\*, Byron Boots, and Mustafa Mukadam

**Abstract**—Efficient planning in dynamic and uncertain environments is a fundamental challenge in robotics. In the context of trajectory optimization, the feasibility of paths can change as the environment evolves. Therefore, it can be beneficial to reason about multiple possible paths simultaneously. We build on prior work that considers graph-based trajectories to find solutions in multiple homotopy classes concurrently. Specifically, we extend this previous work to an online setting where the unreachable (in time) part of the graph is pruned and the remaining graph is reoptimized at every time step. As the robot moves within the graph on the path that is most promising, the pruning and reoptimization allows us to retain candidate paths that may become more viable in the future as the environment changes, essentially enabling the robot to dynamically switch between numerous homotopy classes. We compare our approach against prior work without the homotopy switching capability and show improved performance across several metrics in simulation with a 2D robot in multiple dynamic environments under noisy measurements and execution.

## I. INTRODUCTION & RELATED WORK

Motion planning is a core problem for robotic systems that must successfully navigate their surroundings. In uncertain and dynamic environments, the planning problem becomes especially challenging since a previously feasible solution can quickly become infeasible and must be recomputed. Broadly speaking, motion planning approaches can be grouped into the categories of search, sampling, and optimization.

Search-based algorithms, such as A\* [1] guarantee completeness and optimality, but quickly succumb to the curse of dimensionality as discretization of the environment is necessary. Sampling-based algorithms, including PRM [2] and RRT [3], are probabilistically complete, generalize better to higher dimensional problems, and have been extended to provide (asymptotic) optimality [4]. While these methods are well suited to solve complex problems in environments like 2D mazes, in practice they tend to be very computationally expensive for high dimensional systems and require additional post-processing to smooth the solutions. Conversely, trajectory optimization algorithms initialize a trajectory from start to goal state and then optimize the path by minimizing some cost function [5], [6]. GPMP2 [7] represents smooth, continuous trajectories as samples from a Gaussian Process (GP) and solves the motion planning problem by performing probabilistic inference on a factor graph. While these

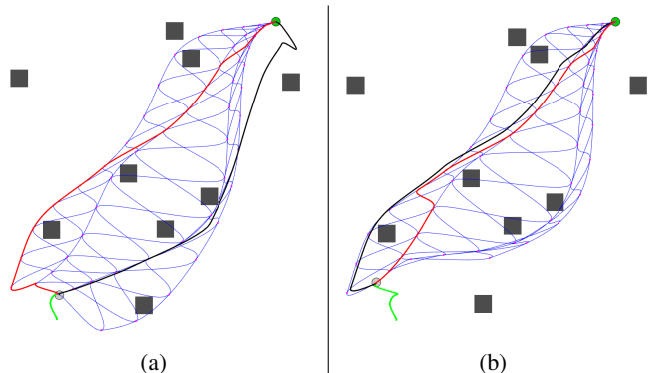


Fig. 1: POSH being used in a dynamic environment where the robot (gray circle) is tasked to reach the goal (green circle) while avoiding obstacles (gray squares). At some time step  $i$  In (a) the robot has traveled along the green path when POSH returns a new best (lowest-cost and feasible) trajectory (red). This represents a homotopy switch from the previous best trajectory (black). It executes the first step of this new trajectory and then at the next time step  $i + 1$  in (b) POSH again returns a new best trajectory (red) that is in a different homotopy class.

methods are much faster for high dimensional systems, they are sensitive to initialization and can get stuck in poor local optima. In practice, straight-line initializations have been shown to work well and random restarts are performed if that fails [5], [6]. Initializing with a feasible solution from a sampling-based planner is also an option [5], but this incurs extra computational burden.

Recently, hybrid approaches to planning have attempted to combine the complimentary benefits of different planning schemes. For example, BIT\* [8] combines graph-based and sample-based planning by using a heuristic function to guide a series of random geometric graphs towards the goal state. Its successor, RABIT\* [9] performs BIT\* while running trajectory optimization when constructing edges between samples, thus reducing sampling complexity by decreasing the number of samples discarded. GPMP-GRAPH [10], an extension of GPMP2 [11], constructs a graph of interconnected trajectory initializations, similar to the one shown in Fig. 1, on a factor graph that when optimized can simultaneously evaluate an exponential number of initializations. After optimization, any path through this graph provides a feasible trajectory. Thus, it has the ability to find solutions in multiple unique homotopy-classes all at once.

While these planning approaches are typically employed in static environments, a number of algorithms [12] have been proposed to handle navigating dynamic environments. Life-long Planning A\* [13], D\* lite [14] and comparable online

\*Equal contribution.

The authors are with the Robot Learning Lab, Georgia Institute of Technology, USA. Email: {kkolar3, schintalapudi}@gatech.edu.

planning algorithms outperform repeatedly replanning from scratch by incrementally finding shortest paths on a graph with changing edge costs. RRT-X [15] an extension of RRT is suited to environments with moving obstacles and provides comparable runtime performance. RAMP [16] draws from evolutionary computation by maintaining a population of trajectories and evaluating their respective quality (fitness) with respect to a cost function and has been shown to discover and exploit new homotopies. Since solutions can go in and out of feasibility as the environment changes access to solutions in multiple homotopy classes is beneficial. However, there exists a gap in current research with respect to solving the online planning problem by leveraging trajectories across different homotopy classes.

In this work, we present a novel trajectory optimization algorithm, Planning Online by Switching Homotopies (POSH), to handle such scenarios. We build on GPMP-GRAPH [10] where multiple trajectories are inter-connected and represented as a factor graph upon which probabilistic inference is performed to optimize the entire graph. However, instead of retaining the same optimized trajectory from the initial time step for execution as GPMP-GRAPH does, POSH maintains and updates the entire graph at every time step. Specifically, at any time step the graph is pruned to remove unreachable (in time) states and is then reoptimized considering changes in the environment to find the new optimal trajectory. This grants our algorithm the unique ability to dynamically switch between different homotopy classes as illustrated in Fig. 1 and allows it to better contend with dynamic environments as demonstrated by our experiments.

## II. BACKGROUND

In this section we review GPMP-GRAPH, closely following the explanation of Huang et al. [10], as well as other concepts, such as homotopy classes and their application to motion planning, necessary to explain our approach.

### A. GPMP-GRAPH

GPMP-GRAPH is a graph-based trajectory optimization algorithm designed to address the local minima problem by evaluating an exponential number of initializations simultaneously and, as a consequence, finding solutions in multiple homotopy classes. The graph is constructed as a series of chains between the start and goal state, with connections between the chains. Optimization happens over the entire graph, yielding improvements in planning time over sequentially optimizing a single chain with different initializations [10].

GPMP-GRAPH treats the motion planning problem as probabilistic inference. Given desired events  $e$ , it finds the maximum a posteriori (MAP) trajectory  $x^*$

$$x^* = \arg \max_x p(x|e) = \arg \max_x p(x)l(x;e)$$

where  $p(x)$  is the prior distribution over trajectories and  $l(x;e)$  is the likelihood.

Continuous-time trajectories are represented as samples from a vector-valued Gaussian process (GP)  $x(t) \sim$

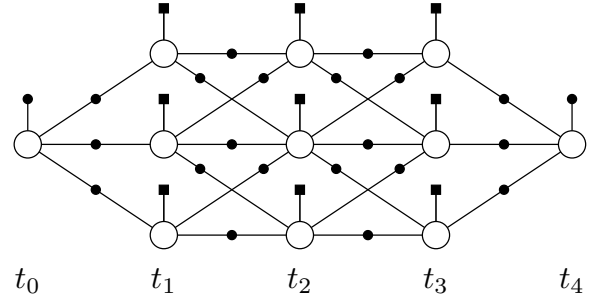


Fig. 2: Factor Graph consisting of three Gauss-Markov Chains as illustrated in [10]. The white circles represent support states. Black squares are collision factors and black circles are GP prior factors. Interpolated collision factors are omitted for clarity.

$\mathcal{GP}(\mu(t), \mathcal{K}(t, t'))$  where  $\mu(t)$  is a vector-valued mean function and  $\mathcal{K}(t, t')$  is a matrix-valued covariance function. Following the definition of GP, trajectories can be parameterized at discrete times  $t = \{t_0, \dots, t_n\}$  to be jointly Gaussian distributed

$$x = [x_0, \dots, x_n]^T \sim \mathcal{N}(\mu, \mathcal{K})$$

where:

$$\mu = [\mu(t_0), \dots, \mu(t_n)], \quad \mathcal{K} = [\mathcal{K}(t_i, t_j)]_{i,j,0 \leq i,j \leq n}$$

The values  $x_0, \dots, x_n$  are referred to as *support states* and correspond to  $x(0), \dots, x(n)$ . This defines a prior distribution over trajectories

$$p(x) \propto \exp \left\{ -\frac{1}{2} \|x - \mu\|_{\mathcal{K}}^2 \right\}$$

In practice, this distribution can be constructed from robot dynamics, for example, modeled as a linear time-varying stochastic differential equation (LTV-SDE)

$$\dot{x}(t) = A(t)x(t) + u(t) + F(t)w(t)$$

where  $A, F$  are system matrices,  $u$  is a known control input, and  $w(t) \propto \mathcal{GP}(0, Q_c \delta(t - t'))$  is a white noise process with  $Q_c$  being the power spectral density matrix of the system and  $\delta$  being Dirac delta function. The first- and second-order moments of the solution to this SDE yield the mean and covariance of the GP, respectively. The prior penalizes the trajectory from deviations from the mean to maintain smoothness in the solutions. The likelihood  $l(x;e)$ , handles costs like obstacle avoidance and any other motion constraints.

The posterior distribution  $p(x|e)$  can be represented as a *factor graph*. A factor graph is a bipartite graph  $G = \{\Theta, F, E\}$ , where  $\Theta$  is a set of variables,  $F$  is a set of factors, and  $E$  is the set of edges connecting  $F$  to  $\Theta$ . Then the posterior can be factorized as

$$p(x|e) = \prod_{t_i} f_i^{gp}(x_i, x_{i+1}) f_i^{obs}(x_i) \prod_{\tau} f_{i,\tau}^{intp}(x_i, x_{i+1})$$

where  $f_i^{gp}$  is a binary factor connecting consecutive states,  $f_i^{obs}$  is a unary collision likelihood factor, and  $f_{i,\tau}^{intp}$  is a

collision likelihood factor for a state at  $\tau$  between consecutive support states. This factor graph has a sparse structure which can be exploited for rapidly solving the optimization problem [7].

So far this is identical to GPMP2 [7], however GPMP-GRAPH constructs such a factor graph with multiple initial trajectories (chains), each with its own set of support states. Adjacent chains have interconnected support states with a GP prior factor and interpolated factors between two states from two different chains, allowing the robot to switch between chains. An example graph construction with three chains and interconnections is shown in Fig. 2. The graph's variables (corresponding to trajectory waypoints) are optimized using the Levenberg-Marquardt algorithm to maximize their probability given the desired events, such as being collision free and conforming to motion constraints. Then, from the optimized graph we can find the trajectory with the lowest cost or the trajectory that conforms to any predefined homotopy constraints set by the user. For more details please refer to [10].

GPMP-GRAPH suffers in practical performance because it is unable to exploit the different homotopy classes it identifies when the environment changes. By constructing a factor graph with multiple chains and then allowing those chains to converge to low-cost solutions in different homotopy-classes, the algorithm has many feasible trajectories available to it. However, dynamic obstacles may cause an initially good trajectory to decrease in quality if the obstacle moves in the way of the trajectory. Only optimizing the chain that the robot is on at that time may not allow the robot to switch homotopies, leading to collisions with the moving obstacle or inefficient solutions.

### B. Homotopy Classes

GPMP-GRAPH improves upon its predecessor GPMP2, by initially considering multiple homotopy classes when optimizing over different initial trajectories in parallel. Two continuous functions over the real numbers  $x_1, x_2 : \mathbb{R} \rightarrow \mathbb{R}$  are said to be homotopically equivalent if there exists a continuous function

$$h : \mathbb{R} \times [0, 1] \rightarrow \mathbb{R}$$

such that  $h(t, 0) = x_1(t)$  and  $h(t, 1) = x_2(t), t \in \mathbb{R}$ . In the context of motion planning, it is said that two trajectories (with a fixed start and end state) belong to the same homotopy class if one trajectory can be *continuously deformed* into the other without intersecting an obstacle [17]. This deformation is given by the homotopy function  $h$ .

We use the concept of *h-signature* to identify the frequency at which the robot changes the homotopy class of its trajectory. An h-signature is a unique identifier for a homotopy class in an environment. Two trajectories have the same h-signature if and only if they belong to the same homotopy class [18]. The h-signature for a trajectory is determined by extending a vertical ray upwards towards  $y = +\infty$  from the center of each obstacle and assigning a letter  $t_k$  to obstacle  $k$ . We iterate through each point in the trajectory  $x$  from

start to goal and check at every time step  $t$  if the trajectory crosses a ray between  $t$  and  $t - 1$ . If the trajectory crosses obstacle  $k$  from left to right we append  $t_k$  to the h-signature of the trajectory and if the trajectory crosses obstacle  $k$  from right to left we append  $\bar{t}_k$  to the trajectory. The final h-signature is produced by reducing this signature by removing all instances of  $t_k t_k$  or  $\bar{t}_k \bar{t}_k$  in the signature. [18] proves that this procedure yields a signature that uniquely corresponds to a homotopy class in the environment.

By identifying trajectories in different homotopy classes, POSH is finding solutions to the problem that do not deform into each other. This is useful because trajectory-optimization methods like GPMP-GRAPH typically will not deform a solution through an obstacle, preferring to get caught in a local minima in the current homotopy class. By designing POSH such that trajectories in multiple homotopy-classes are maintained, the planner will better be able to adapt to changes in the environment that render the current trajectory suboptimal.

## III. PLANNING ONLINE BY SWITCHING HOMOTOPIES

We begin by motivating POSH with our key insight that switching homotopy classes is essential to navigating changing environments. Then we present our algorithm and its implementation details.

### A. Motivation

In Fig. 5 we see GPMP-GRAPH try to navigate through a 2-Dimensional environment with a moving obstacle. While at time step 1 the robot seems to have found a high quality trajectory, changes in the environment start to degrade that trajectory. Namely, reoptimizing a single chain means that the robot will consider only its current homotopy-class, as deforming the trajectory through an obstacle would lead to an increase in trajectory cost. As a consequence, the trajectory is pushed further into a corner as the obstacle approaches, leading to multiple collisions in the trajectory. Intuitively, it seems reasonable to travel along the left of the moving obstacle if it is moving closer, however such a solution would be in a different homotopy class. For this reason, we should maintain and reoptimize trajectories in both the homotopy classes, the one to the right of the obstacle and the one to the left of the obstacle. Then, at every time step the robot can replan a solution to the goal state using the reoptimized chains of the factor graph. As the obstacle moves closer, the low-quality trajectory to the right of the obstacle will suffer a higher cost than a trajectory that moves to the left of the obstacle. Thus, the robot will now respond to this change in trajectory quality by changing the homotopy class mid-execution to avoid collision.

### B. Algorithm

POSH is summarized by Algorithm 1. The initial obstacle positions are obtained to create the initial SDF as shown by line 1. In line 2, a factor graph is constructed given initial robot states, a signed distance field (SDF) of the environment, and parameters like smoothness prior (desired

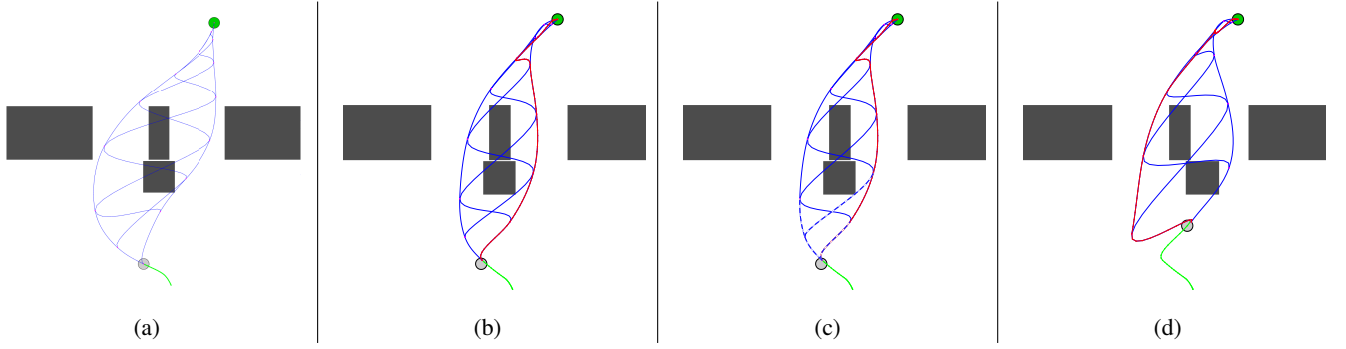


Fig. 3: In (a), the gray circle and green circle represent the robot and goal state respectively. The graph is warm-started from its previous state. In (b), the graph is reoptimized and then A\* is used to find a low-cost trajectory (red) on the graph. In (c), edges that will no longer be traversable (dashed blue lines) after executing the first control are pruned. Finally in (d), the robot executes its first control as the environment changes. We see that POSH finds a trajectory in a new homotopy class (red) after reoptimizing.

---

**Algorithm 1** POSH

---

```

1:  $SDF \leftarrow \text{initial\_obstacle\_poses}$ 
2:  $\text{factor\_graph} \leftarrow \{\text{parameters}, \text{robot\_states}, SDF\}$ 
3: for  $t = 1, \dots, T$  do
4:    $\text{factor\_graph.optimize}()$ 
5:    $x \leftarrow A^*(\text{factor\_graph})$ 
6:    $\text{factor\_graph.prune\_unreachable}()$ 
7:    $\text{execute\_step}(x)$ 
8:    $SDF.\text{update}(\text{obstacle\_poses})$ 
9: end for

```

---

trade-off between smoothness and collision avoidance) and ratio of interconnections (ranges from 0 i.e. no connections to 1 i.e. every state is interconnected). Given a number of desired trajectories (chains) to be interconnected and a number of time steps, the locations of the initial support states are computed by constructing a series of ellipses with the major radius stretching from the start state to the goal state and varying minor radii as shown in Fig. 4a. Based on the desired ratio of interconnections, states in spatially nearby chains are then interconnected to allow switching between them (see Fig. 4b and Fig. 4c). In line 4, given the initial graph and SDF, POSH employs the Levenberg-Marquardt algorithm to optimize the factor graph.

Up to this step, POSH has executed the same steps as GPMP-GRAPH would in solving the motion planning problem. Following this, GPMP-GRAPH simply executes the steps in the most feasible trajectory initially obtained and reoptimize this same trajectory at every time step until the goal configuration is reached. However, as discussed earlier this is insufficient to successfully plan in a stochastic, dynamic environment. POSH reoptimizes the entire graph with an updated SDF and re-calculates the most feasible trajectory. These steps, shown in lines 4-8 in Algorithm 1, are executed at every time step until the goal state is reached.

We use the example in Fig. 3 to illustrates the process POSH follows. Fig. 3a shows the graph before an iteration of optimization. The robot is the gray circle in the bottom left and the green circle in the top right is the goal state. The graph’s current state represents the optimal graph configura-

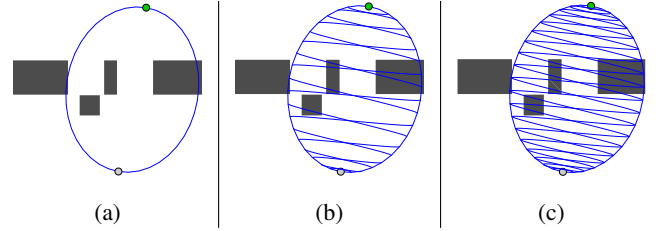


Fig. 4: Possible graph initializations with (a) no connection between chains, (b) connections between chains at every other state, and (c) fully connected chains. The robot is the gray circle and the goal state is the green circle.

tion found given the SDF at the last time step. In Fig. 3b, POSH re-optimizes the factor graph based on the updated SDF. Then, as shown in line 5 of Algorithm 1, A\* is run on this updated, optimized factor graph to return the lowest cost trajectory (plotted in red). Because A\* is complete and optimal, and the graph connects the start and goal states, it is guaranteed that the returned solution will be the lowest cost solution out of possible feasible trajectories encoded in the graph.

This is followed by line 6, in which POSH runs depth first search (DFS) to prune away a subgraph of the factor graph to only maintain the reachable states. Given the current state as the root and the goal state, DFS determines which support states are unreachable in time and are therefore irrelevant. In Fig. 3c, the pruned away subgraph is shown with dashed blue lines. By executing the first step (line 7 of Algorithm 1) in the trajectory returned by A\* in Fig. 3c, the robot arrives at its new position depicted in Fig. 3d. At the next time step, the obstacle moves closer to the trajectory the robot was planning to take as Fig. 3d shows. Then in line 8 the SDF is updated given changes in the environment. After running another iteration of optimization and A\*, the robot now identifies a new trajectory in a different homotopy class that is lower cost and feasible. By doing so, POSH avoids the local minima problem that GPMP-GRAPH is unable to contend with as seen in Fig. 5, resulting in a collision. On the other hand, POSH successfully adapts to the dynamic environment by switching homotopies. This process repeats until the robot has reached the goal.

### C. Implementation Details

POSH is implemented by building on the GTSAM [19] and GPMP2 library. The factor graph is optimized using the Levenberg-Marquardt algorithm initialized with  $\lambda = 10^{-5}$ . In order to successfully plan in a stochastic, dynamic environment, POSH must optimize the factor graph based on an updated SDF and determine the most feasible trajectory. Only the first step is executed as this trajectory might not be feasible for future configurations of the environment. To replan efficiently at the next time step, the relevant portions of the optimized factor graph must be retained so that the graph is not reconstructed from scratch. POSH accomplishes this by generating a spanning tree of the graph rooted at the first state in the most feasible trajectory, before navigating to it. The subgraph of the factor graph that does not belong to this spanning tree is pruned away as the states in this subgraph become unreachable in the next time step. Additionally, to replan quicker, the next optimization of the graph is warm-started with the state of the previously initialized factor graph. This results in Levenberg-Marquardt taking less time to optimize the factor graph, since the initialization of the graph was a local minima of the cost function at the last time step, assuming the environment did not drastically change between time steps.

## IV. RESULTS

We benchmark POSH against GPMP2 and GPMP-GRAPH, two trajectory optimization methods within the same family on multiple 2D dynamic environments<sup>1</sup>. GPMP2 and GPMP-GRAPH are set up as a single chain and interconnected multi-chain graphs respectively. At each time step the GPMP2 chain is optimized and a step is taken on the optimized path. Then, the past state is pruned, the noisy state measurement is added, and the chain is reoptimized for the next time step. Since GPMP-GRAPH is an offline batch method, to keep comparisons fair, we will optimize the full multi-chain graph at the first time step to get an optimal sequence of states (a chain), while the other states are pruned away. From then on it is treated as a GPMP2 chain and is optimized accordingly at every time step. In contrast, POSH will continue to keep a pruned graph after every time step to be reoptimized.

These algorithms' performance are tested on two simulated 2D datasets, a dynamic forest and narrow passageway environment. In both environments, the holonomic robot, starting from some initial state, is tasked with reaching the goal while avoiding obstacles. The robot is also subjected to stochastic executions and noisy measurements i.e. the robot's position at every time step is perturbed to simulate execution noise and localization noise is injected into state measurements respectively. Sections IV-A and IV-B will describe the narrow passageway and dynamic forest environments respectively as well as analyze the algorithms' performance in each. We record the following metrics averaged across

TABLE I: Benchmark results on narrow passage dataset.

| Metric                  | POSH        | GPMP-GRAPH | GPMP2 |
|-------------------------|-------------|------------|-------|
| Success Rate (%)        | <b>100</b>  | 10         | 10    |
| Collision Intensity (%) | <b>0</b>    | 6.5        | 7.5   |
| Distance (m)            | <b>26.7</b> | 31.6       | 33.8  |
| Homotopy Switches       | <b>1.2</b>  | 0.2        | 1.1   |

multiple runs in each environment: (i) success rate to quantify the runs where the robot gets to the goal without any collisions, (ii) collision intensity to quantify the percentage of the trajectory spent in collisions when they occur, (iii) total distance traveled indicative of smoothness in the path, and (iv) total number of homotopy switches made from reoptimizing at each time step.

For all algorithms, the safety distance,  $\epsilon$ , and obstacle cost weight,  $\sigma_{obs}$  (represents the trade-off between a trajectory smoothness and collision avoidance), are kept constant. The execution prior (initial state) and localization noise levels are all kept constant as well. To account for stochasticity, each algorithm's performance was averaged over 10 Monte-Carlo runs for each experiment. One narrow passageway environment and 13 dynamic forest environments were used to test the algorithms' performance. In the case of GPMP-GRAPH and POSH, hyperparameters like, the number of chains  $N_I$ , the ratio of interconnections between chains  $R_I$ , and GP prior noise applied to the interconnections between chains  $Q_I$ , are tuned to give good performance for the respective algorithm.

### A. Dynamic Narrow Passageway Benchmark

The dynamic narrow passageway consists of an obstacle moving back-and-forth with constant velocity in a 1D motion (left to right then right to left) in front of a narrow passageway that contains two entrances as shown in Fig. 5. As this obstacle moves in front of the passageway it will alternately block one of the entrances. This environment is used to test the algorithms' online planning capabilities in a controlled setting. As we argued previously the ability to switch homotopies will become important in such a scenario.

The results are summarized in Table I. POSH is able to successfully plan 100% in narrow passage environment, while executing on average 1-2 homotopy switches to explore the both passage entrances. As a result, it also minimizes distance traveled due to this early, prudent homotopy switch. On the other hand, GPMP2 and GPMP-GRAPH are largely unsuccessful at solving this problem as they fail to successfully find collision free paths. While at first, the optimal path from start to goal is obvious for GPMP2, the moving obstacle quickly eliminates this as a viable path, forcing the algorithms' to improvise or fail. They get stuck in local minima as seen in Fig. 5 where the object's lateral motion continues to push the trajectory to the side until finding a collision free path is no longer possible under the given temporal and physical constraints. GPMP-GRAPH's initial solution that spans multiple chains also succumbs to the same issue since it is reduced to a single chain graph

<sup>1</sup>A video of the experiments is available at <https://youtu.be/FtUI9VR3iWI>

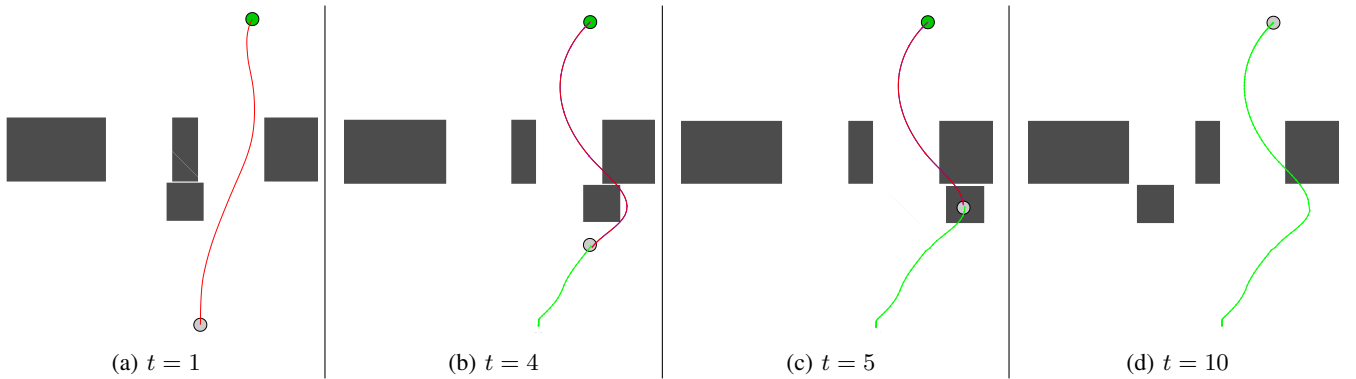


Fig. 5: GPMP-GRAPH in (a) finds the lowest-cost trajectory (red) from the initial graph construction of 2 chains. It optimizes and follows this path at each time step, shown in green. In (b)-(c) the optimized path (red) continues to get pushed laterally due to the moving object, forcing the robot to collide with the obstacle. The final trajectory in (d) shows the robot colliding with the passageway as a result of being unable to successfully plan around the moving obstacle.

after the first time step. On the other hand, Fig. 3 shows that POSH contends with this local minima problem by utilizing a multi-chain factor graph to explore a path in the other homotopy class once it becomes more viable than the previously desirable homotopy. It is important to note that the homotopy switches made by GPMP-GRAPH and GPMP2 are a result of the dynamic environment changing the optimized path's *h-signature* rather than an intended algorithmic or strategic homotopy switch.

### B. Dynamic Forest Benchmark

The dynamic forest environment consists of numerous objects exhibiting random 2D motion. This environment is simulated given a number of obstacles, their initial state and a random bounded acceleration applied at each time step. The dynamic forest is shown in Fig. 1. It is employed to benchmark the ability of the algorithms to contend with a harder replanning problem.

The dynamic forest benchmark results are shown in Table II. Our algorithm generates paths that are more successful and when it fails it spends less time on average in collision. Again, this can be attributed to POSH constantly replanning and reoptimizing the multi-chain factor graph instead of sticking with a single chain factor graph, allowing for online homotopy switching. This process can be seen in Fig. 6 and Fig. 1. Since there are many more paths for POSH to consider, it returns a feasible path more often than the two other algorithms. In Fig. 1, the black line represents the trajectory considered best at the previous time step while the red line represents the trajectory considered best at the current time step. Fig. 1a signifies a homotopy switch as the red and black lines cannot be continuously deformed into each other without colliding with obstacles. Fig. 1b shows the robot executing the first step of the red path from Fig. 1a while again considering another homotopy switch.

Note that GPMP2's on average shorter distance traveled can be attributed to the straight line initialization that characterizes the region around the local minima, but this also results in a greater collision intensity and a much lower success rate.

TABLE II: Benchmark results on dynamic forest dataset.

| Metric                             | POSH         | GPMP-GRAPH | GPMP2         |
|------------------------------------|--------------|------------|---------------|
| Success Rate (%)                   | <b>29.23</b> | 11.92      | 8.46          |
| Collision Intensity (%)            | <b>1.9</b>   | 5.1        | 8.3           |
| Distance (m)                       | 59.05        | 66.52      | <b>53.41</b>  |
| Homotopy Switches                  | <b>5.15</b>  | 4.23       | 3.54          |
| Avg. Comp. Time (s) at $t = 0$     | 0.221        | 0.020      | <b>0.0197</b> |
| Avg. Comp. Time (s) at any $t > 0$ | 0.077        | 0.029      | <b>0.028</b>  |

TABLE III: Performance of POSH with varying number of chains on dynamic forest dataset.

| Metric                  | $N_I=2$     | $N_I=4$      | $N_I=6$ |
|-------------------------|-------------|--------------|---------|
| Success Rate (%)        | 22.31       | <b>29.23</b> | 18.85   |
| Collision Intensity (%) | <b>1.9</b>  | <b>1.9</b>   | 2.2     |
| Distance (m)            | 59.1        | <b>59.0</b>  | 60.6    |
| Homotopy Switches       | <b>5.23</b> | 5.15         | 4.38    |

We also analyze the performance of POSH on factor graphs with different numbers of chains,  $N_I = 2$ ,  $N_I = 4$ , and  $N_I = 6$ . The results in Table III indicate that there is a sweet spot in the middle (at  $N_I = 4$ ) to get the best performance. An example with four chains is visualized in Fig. 6. The two chain factor graph will have fallen into two of the four homotopy classes that the four chain factor graph will have, thus exhibiting similar metrics. The six chain factor graph exhibits a reduction in performance due to chains collapsing into redundant homotopy classes as a result of  $Q_I$  and the number of time steps making the optimization problem too constrained. It is also important to consider the algorithmic computation time when deciding on the number of chains to include in the factor graph as increasing number of chains increases computation time. Ideally the computation time should be smaller than the time length of the trajectory between two time steps ( $\delta t$ ) for POSH to be applicable online, which was indeed the case in our experiments. Table II shows the average computation times for the first iteration at  $t = 0$  and for any other iteration at  $t > 0$ , where  $\delta t = 0.5s$ .



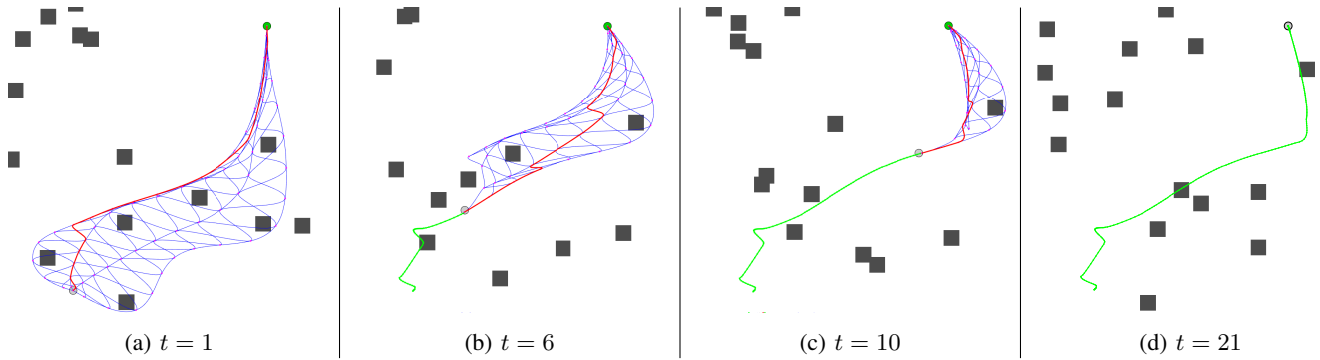


Fig. 6: POSH successfully returns a collision free trajectory utilizing 4 chains. At each consecutive time-step  $t$ , the robot (gray circle) prunes away unreachable portion of the graph and reoptimizes the remaining graph until it reaches the goal state (green circle). The path the robot has traveled so far is shown green and the lowest-cost trajectory going forward is shown in red.

## V. DISCUSSION

In this paper, we introduced POSH, a novel online graph-based trajectory optimization planning algorithm. It builds on recent inference based trajectory optimization techniques to add capability to plan online and dynamically switch homotopies. We found that POSH can leverage knowledge of distinct trajectories across different homotopy classes to dynamically select a low cost solution in light of new information about the environment. In dynamic environments with moving obstacles, previous trajectory-optimization approaches are collision-prone as they become stuck in a local minima. Rather than committing to a low-quality trajectory in a local minima, POSH successfully generates smooth and collision-free trajectories by maintaining and optimizing multiple interconnected solutions to the planning problem.

Our work is currently limited to a finite time horizon setting i.e. the number of discretized time steps we are planning over is fixed. For long range tasks in dynamic environments, occasionally a receding horizon formulation is more beneficial to avoid wasting computation on latter portions of the plan far ahead in the future. Extending our approach to such settings can further improve its capability and applicability. POSH (and GPMP-GRAPH) can sometimes suffer from different trajectories collapsing into the same homotopy class. To contend with this problem it will be beneficial to further investigate principled ways of integrating sampling strategies for new support states such that POSH not only prunes unreachable states, but also add new states at every time step to allow for diversity and exploration.

## REFERENCES

- [1] 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*, vol. 4, no. 2, pp. 100–107, July 1968.
- [2] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.
- [3] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2, IEEE, 2000, pp. 995–1001.
- [4] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, 2010.
- [5] 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*, vol. 32, no. 9-10, pp. 1164–1193, 2013.
- [6] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.
- [7] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research (IJRR)*, 2018.
- [8] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit\*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3067–3074.
- [9] S. Choudhury, J. Gammell, T. Barfoot, and S. Srinivasa, "Regionally accelerated batch informed trees (RABIT\*): A framework to integrate local information into optimal path planning," in *Proceedings of the 2016 IEEE Conference on Robotics and Automation (ICRA)*, 2016.
- [10] E. Huang, M. Mukadam, Z. Liu, and B. Boots, "Motion planning with graph-based trajectories and Gaussian process inference," in *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.
- [11] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using Gaussian processes and factor graphs," in *Proceedings of Robotics: Science and Systems (RSS)*, 2016.
- [12] C. Park, J. Pan, and D. Manocha, "Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments," in *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 06 2012.
- [13] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning a," *Artificial Intelligence*, vol. 155, no. 1, pp. 93 – 146, 2004.
- [14] X. Sun, W. Yeoh, and S. Koenig, "Moving target d\* lite," in *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, 2010, pp. 67–74.
- [15] M. W. Otte and E. Frazzoli, "Rrtx: Real-time motion planning/replanning for environments with unpredictable obstacles," in *WAFR*, 2014.
- [16] J. Vannoy and J. Xiao, "Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environments with unforeseen changes," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1199–1212, Oct 2008.
- [17] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Autonomous Robots*, vol. 33, no. 3, pp. 273–290, 2012.
- [18] D. Grigoriev and A. Slissenko, "Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane," in *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC*, 01 1998.
- [19] F. Dellaert, "Factor graphs and GTSAM: a hands-on introduction," Georgia Tech Technical Report, GT-RIM-CP&R-2012-002, Tech. Rep., 2012.