

Joint Sampling and Trajectory Optimization over Graphs for Online Motion Planning

Kalyan Vasudev Alwala and Mustafa Mukadam

Facebook AI Research

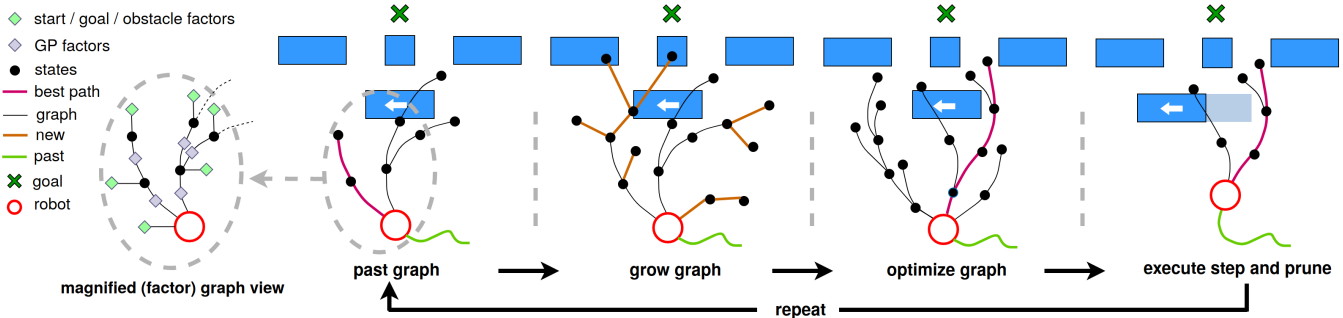


Fig. 1: One iteration of our approach illustrating its ability to track and switch between possible plans in an online dynamic setting.

Abstract—Among the most prevalent motion planning techniques, sampling and trajectory optimization have emerged successful due to their ability to handle tight constraints and high-dimensional systems, respectively. However, limitations in sampling in higher dimensions and local minima issues in optimization have hindered their ability to excel beyond static scenes in offline settings. Here we consider *highly* dynamic environments with long horizons that necessitate a fast on-line solution. We present a unified approach that leverages the complementary strengths of sampling and optimization, and interleaves them both in a manner that is well suited to this challenging problem. With benchmarks in multiple synthetic and realistic simulated environments, we show that our approach performs significantly better on various metrics against baselines that employ either only sampling or only optimization. Project page: <https://sites.google.com/view/jstplanner>

I. INTRODUCTION AND RELATED WORK

Many real world environments, like warehouses, hospitals, and roads, where we want our robots to operate and move about, necessitate that robots compute their motions very quickly while satisfying feasibility. We study the challenging regime of large, dense, and highly dynamic environments like these and propose a method for online motion planning in such domains.

Early research in planning with search based methods [1], [2] and their anytime variants [3], [4] work well on grids or graphs, the domains for which they were originally developed. These methods don't scale well to high-dimensional continuous domains and therefore have seen limited success in online or dynamic settings [5]–[7]. Current successful and dominant strategies in motion planning can be grouped into sampling and trajectory optimization based approaches. Sampling methods [8]–[10] sample the configuration space of the robot to build feasible (commonly collision free) trees or graphs that (sub)optimally connect the start and goal. Follow up work includes optimal variants [11], task relevant extensions like handling kinodynamic constraints [12], [13],

and improving efficiency [14]–[17]. They can explore the state space given sufficient time or samples and thus are well suited to low dimensional problems with tight constraints like corridors and doorways. On the other hand, optimization methods [18]–[23] optimize an initial trajectory based on an objective function that captures some notion of optimality and feasibility. These methods are able to exploit local information to quickly find solutions and are better suited to high dimensional problems in relatively sparser settings, such as a robot arm reaching for an object on a table. Overall, both sampling and optimization notably excel at offline planning in static environments.

However their limitations become pronounced in large, dense, and dynamic settings that we consider in this work. Extending common sampling based methods to such scenarios is challenging since many samples and edges can become infeasible as the environment changes. Thus progress has been scarce for sampling methods in online [24] and dynamic settings [5], [25]–[28]. A potential course of action in an online setting is to plan as quickly as possible and treat every time step as a new planning problem. This strategy is often utilized by optimization based methods and is analogous to model predictive control [29], [30]. However, optimization can often get stuck in local minima as it is able to focus in the locality of only one hypothesis at a time. Multiple initializations (which may be tricky to specify) can be tried, but by incurring the penalty of added computation, linear in the number of initializations. Thus past work in employing optimization for online and dynamic problems [31]–[34] have been limited to much sparser environments.

In parallel, early planning work has also researched reactive methods [35]–[39] that from a trajectory optimization perspective solve for a trajectory of one time step length. While this makes them extremely fast in practice, it also further exacerbates their local minima problem. Other hybrid

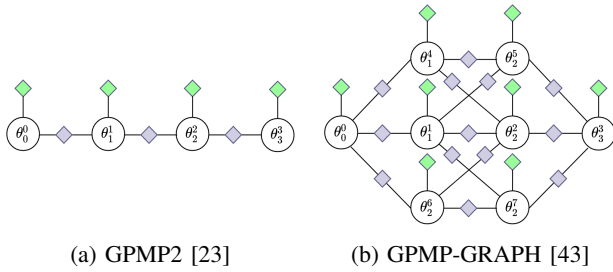


Fig. 2: Example factor graphs. Random variables are white circles and factors are colored shapes.

methods, for example alternating between solving a sampling and an optimization problem [40], a selector that picks an initialization (straight line, random, or sample based) and an optimizer [41], and using optimization in generating the edges in a sampling method [42], tend to carry over the limitations of their underlying approaches and have continued the trend in tackling only static offline settings.

A constantly evolving solution landscape makes the problem of online planning in highly dynamic environments challenging. Regions that were previously feasible can quickly become infeasible and vice versa. In order to succeed in such settings, a planner must ideally (i) be able to continuously keep track of multiple possible hypothesis and switch between them as desired, (ii) have the ability to grow its hypothesis space in search of new possible solutions while effectively pruning less promising options, and (iii) be able to do all of this in a limited computational budget.

Our main contribution is an approach, JoInt Sampling and Trajectory optimization over graphs (JIST), that combines the complimentary strengths of exploration from sampling methods and of exploitation from optimization methods. Specifically, we employ factor graphs as the backbone data structure and extend prior work [22] (that uses chain-like graphs) to instead build more complex tree-like graphs via sampling before optimization and then pruning the graph after executing a step from the optimized solution. Repeating this process online as illustrated in Fig. 1, allows our method to achieve the three desirable properties discussed above i.e. track and switch between multiple possible plans efficiently. In various large, dense, and dynamic simulation environments with planar robots and manipulators, we show that our approach significantly outperforms a pure sampling or optimization strategy, across several metrics. For realism, in our experiments we subject the robots to limited visibility, noisy state measurements, and stochastic actions.

II. THE FACTOR GRAPH BACKBONE

We begin by briefly reviewing the GPMP2 motion planner [23] and its factor graph [44] based trajectory optimization framework. Then we discuss what makes the underlying factor graph formulation a suitable choice to serve as the backbone for our approach.

The planning problem in GPMP2 is formulated as probabilistic inference on a factor graph and then solved via nonlinear least squares through the duality between inference and optimization [45]. This enables using factor graphs as a data structure to flexibly represent the problem, while solving

them with more efficient linear algebra tools instead of message passing. Example factor graphs are shown in Fig. 2 where any factor connected to a set of random variables (nodes) represents the cost function involving those variables.

In general, the product of all conditional probabilities f_k specified by the factors gives the posterior distribution of all the random variables $\theta = \{\theta^i\}$ in the graph (i is the id). Likewise, the weighted sum of all cost functions h_k arising from the factors then specify the objective function whose minimization corresponds to finding the mode of the posterior

$$\begin{aligned} \theta^* &= \operatorname{argmax}_{\theta} \prod_k f_k = \operatorname{argmax}_{\theta} \prod_k \exp \left\{ -\frac{1}{2} \|\mathbf{h}_k(\Theta_k)\|_{\Sigma_k}^2 \right\} \\ &= \operatorname{argmin}_{\theta} \sum_k \left\{ \frac{1}{2} \mathbf{h}_k(\Theta_k)^\top \Sigma_k^{-1} \mathbf{h}_k(\Theta_k) \right\}. \end{aligned} \quad (1)$$

Any cost h_k depends on Θ_k a subset of variables and is weighted by the inverse covariance Σ_k^{-1} of the factor. This optimization can be solved with an iterative scheme like Gauss-Newton by linearizing around the current solution and solving the following linear system

$$\left\{ \sum_k \mathbf{J}_k^\top \Sigma_k^{-1} \mathbf{J}_k \right\} \delta \theta = - \sum_k \mathbf{J}_k^\top \mathbf{h}_k(\Theta_k) \quad (2)$$

where Jacobian $\mathbf{J}_k = \partial \mathbf{h}_k / \partial \theta$. Then the subsequent update can be performed with $\theta \leftarrow \theta + \delta \theta$.

In GPMP2 this factor graph takes the form of a chain as shown in Fig. 2(a) that symbolizes the trajectory where any random variable θ_t^i with id i is the state of the robot at some time step t . For instance, this state can be a vector of some d -dimensional configuration position and velocity, $\theta_t^i \in \mathbb{R}^{2d}$. This graph consists of several factors: (i) any two consecutive states in time are connected by a GP (Gaussian Process) factor that specifies the motion model of the robot and encodes the optimality criteria to keep the trajectory smooth, (ii) every state except the first and last also connect to an obstacle factor that ensures that the state remains collision free, and (iii) the first and last state connect to a start and goal factor respectively to constrain the trajectory between start and goal locations. Similarly other factors can be added to the graph (to be part of the objective function) to accommodate other constraints like joint limits, collision avoidance between the states, etc. These and other factors we use in our work are detailed in the Appendix.

The optimization problem in Eq. (1) points out that in relation to an arbitrary factor graph this formulation can be very general and not restricted to chain like graphs that encode only a single trajectory. This idea was leveraged in GPMP-GRAPH [43] to evaluate an interconnected network of trajectories all at once. As illustrated in Fig. 2(b), it builds the factor graph with multiple chains connecting the start and goal. These chains are also connected to each other based on the spatial neighborhoods of their initialization. When this graph is optimized it results in a better exploration of the state space and thus helps find multiple solutions or handles problems with few good local minima. This was later turned into an online approach by POSH [46] where

after taking a step on the desired path from GPMP-GRAPH the unreachable part of the graph is pruned, the environment update is received, and the graph is reoptimized to get a new path. This process is repeated until the end of the graph is reached and allows POSH to switch between different hypothesis paths as and when they became more desirable. However, not having the ability to grow the graph in any way hinders POSH in highly dynamic or long horizon problems. Additionally, the reliance on the GPMP-GRAPH template makes it tedious to define the graph structure and initialize it beyond planar settings.

The factor graph perspective does however offer a flexible way to specify any arbitrary network of trajectories with nodes as states and factors as various cost functions. The emerging sparsity can then be easily exploited to efficiently solve the optimization problem and find the optimal states encoding multiple possible solutions. In order to tackle online planning problems in highly dynamic environments, we will leverage this factor graph structure as the backbone for our approach as it will provide a way to efficiently track and switch between different hypothesis solutions. We discuss how to accomplish this in the next section.

III. COMBINING SAMPLING AND OPTIMIZATION

In order to efficiently plan in highly dynamic environments our key idea is to use a factor graph as the underlying data structure over which we employ a sampling strategy to build and grow the graph at any time step, then optimize, prune and repeat. We explain our approach, Joint Sampling and Trajectory optimization over graphs (JIST) with the help of Algorithm 1 and a toy example shown in Fig. 1.

We begin at an intermediate iteration of JIST as shown in Fig. 1 where the robot is at some current location trying to get to the goal θ_{goal} after having traveled from the start state θ_{start} . It has a factor graph \mathcal{G} carried over from the previous iteration. During the first iteration this graph would be initialized with the start state as shown in line 13 of Algorithm 1. As discussed in the previous section, this graph consists of various factors like GP, obstacle avoidance, goal reaching, etc that capture the motion planning objective and can be customized based on the specific task at hand. The factors we use in our experiments are described in the implementation details and elaborated in the Appendix.

In the first stage we perform exploration to discover new hypothesis solutions. We do this with the *GrowFactorGraph* function in line 15 with an RRT-like random sampling strategy, which also leads to the underlying factor graph’s tree like structure where its root is the current robot state. The critical difference compared to standard RRT [9] is that we do not check the samples or the subsequent edges for collision and instead rely on the factors in the graph and the optimization in the next stage to move them out of possible collision. Not only is this more efficient as it reduces redundant computation, but it also allows the samples to be moved by the optimizer leading to local refinement. Given the online nature and long horizons, another difference is that instead of growing

Algorithm 1 JIST

```

1: function GrowFactorGraph
2:    $SDF \leftarrow env.Observe()$ 
3:   while  $\mathcal{G}.size() < node\_budget$  do
4:      $\theta_{rand} \leftarrow RandomSample()$ 
5:      $\theta_{near} \leftarrow \mathcal{G}.NearestNeighbour(\theta_{rand})$ 
6:      $\theta_{new} \leftarrow ExtendState(\theta_{near}, \theta_{rand})$ 
7:      $vertex \leftarrow \mathcal{G}.addVertex(\theta_{near})$ 
8:      $edge \leftarrow \mathcal{G}.addEdge(\theta_{near}, \theta_{new})$ 
9:      $\mathcal{G}.addFactors(edge, vertex, SDF)$ 
10:  end while
11: end function
12: function Main( $\theta_{start}, \theta_{goal}$ )
13:   $\mathcal{G}.Initialize(\theta_{start})$ 
14:  while not at  $\theta_{goal}$  or terminal do
15:     $\mathcal{G}.GrowFactorGraph()$ 
16:     $\mathcal{G}.Optimize()$ 
17:     $\theta_{next} \leftarrow \mathcal{G}.Search()$ 
18:     $Robot.Execute(\theta_{next})$ 
19:     $\hat{\theta}_{next} \leftarrow Robot.MeasureState()$ 
20:     $\mathcal{G}.PruneUnreachable(\hat{\theta}_{next})$ 
21:  end while
22: end function

```

the graph until we reach the goal, we grow the graph until the number of states in the graph, $\mathcal{G}.size()$ hits a preset number, *node.budget*. *node.budget* implicitly controls the computation needed by our method and can be tuned as a hyperparameter to achieve the best performance. As the graph is grown, factors are added in line 9, including the updated signed distance field (SDF) of the environment observed by the robot which is used for collision avoidance.

In the next stage we exploit the local information in line 16 and optimize the new graph that consists of the old graph and the newly sampled portion. The optimization is efficiently performed with the iterative scheme in Eq. (1)-(2) until convergence to get all the locally optimal states in the graph. This enables our approach to track multiple possible hypothesis paths. Now we can choose the next best step or steps to be executed by searching over the converged graph and leads to a potential switch in the current path compared to previous iteration as shown in Fig. 1. Finally, the (noisy) robot state is measured after (stochastic) execution and the unreachable parts of the graph are pruned. This procedure is repeated during every iteration of JIST until the goal is reached or another terminal condition (like collision) is met.

Thus our method JIST explores via sampling and exploits via efficient optimization and combines the benefits of both by leveraging a factor graph backbone that dynamically grows and shrinks between iterations. This allows it to explore, track, and switch between possible solutions as the environment changes and the solution space evolves. These features are desirable in planning for environment that are large, dense, and highly dynamic. JIST also offers flexibility in choosing any sampling strategy, objectives for the cost, optimization, as well as the method to search over the graph, based on what is relevant to the task and application.

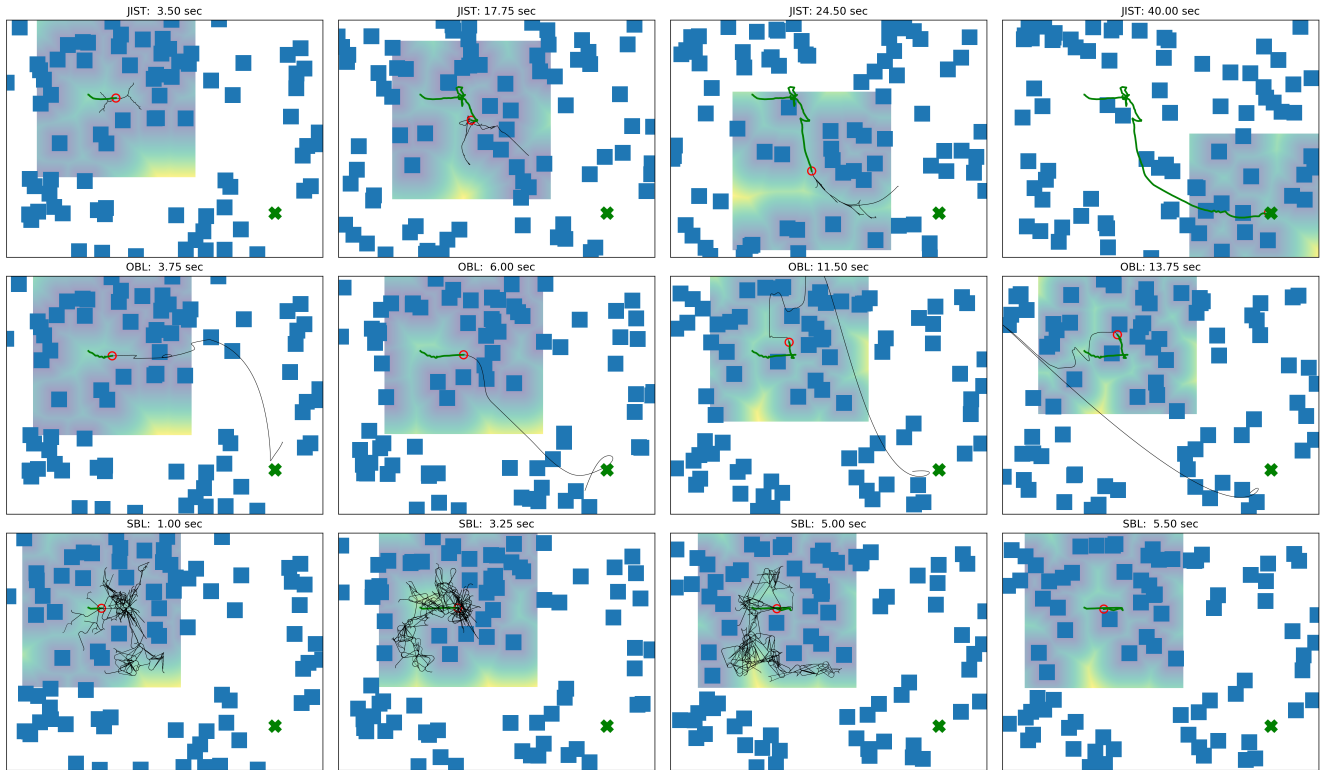


Fig. 3: Same 2D Forest benchmark example (left to right) where JIST (ours) (top row) successfully reaches the goal while OBL (middle row) and SBL (bottom row) terminate in collision.

A. Implementation Details

We implemented JIST using the GTSAM [47] and the GPMP2 [23] libraries. While sampling states and building the factor graph, we add the following factors on the states and between the states that encode the motion planning objectives. We apply a Gaussian factor on the current state of the robot to encode the measurement and anchor it as the root of the tree structured graph. Consecutive states in time are connected by the Gaussian process (GP) factor that encodes smoothness of the trajectory (i.e. the motion model of the robot). For collision avoidance we use unary obstacle factors [23] on all states and binary obstacle factors between consecutive states that use GP interpolation and ensure path safety [23]. For goal reaching we adopt the goal factor from [48] and similarly have a goal factor on every state except the current state which allows the graph to drive all paths toward the goal weighted by how far the robot currently is from the goal. We also use domain specific factors that enforce different constraints like velocity limits, joint limits, self-collision, etc and identify them in the next section where we describe each domain. All factors used in our work are detailed in the Appendix.

For searching over the optimized factor graph and choosing the best path and action to execute, we employ a shortest path search where we pick the action or path leading to the leaf robot state on the factor graph with the lowest cost. The cost of any leaf robot state in the factor graph is computed as a summation of cost of all factors on the path leading up to the leaf robot state normalized by the depth of the leaf robot state to account for branches with different depths.

IV. EVALUATION

We benchmark our approach **JIST** that combines sampling and optimization against a sampling only baseline (**SBL**) and an optimization only baseline (**OBL**), in different highly dynamic environments in the domains of robot navigation and manipulation.

Baselines For the optimization baseline OBL, we setup GPMP2's [23] single chain-like factor graph to operate in a receding horizon fashion that resembles MPC [48], but in state space. This serves as a stand-in for state-of-the-art in fast optimization based planning that can only reason over one hypothesis solution at a time. JIST reduces to OBL without the sampling enabled exploration and the resulting tree-like factor graph that holds multiple potential hypothesis to be optimized. On the other hand, the sampling baseline SBL, is based on dynamic variants of RRT* [28] modified to operate in a receding horizon style to work in large environments (i.e. where start and goal can be far apart) and serves as a stand-in for state-of-the-art in sampling based methods. Any iteration of SBL starts with the previous tree after pruning invalid and unreachable nodes and edges. The new samples are added while rewiring the tree towards the goal and terminates after hitting the node budget return the branch getting closest to the goal. JIST reduces to SBL without running the optimization and instead checking node and edge validity while rewiring the tree. The same cost functions and factors are used in OBL and SBL as used in JIST and all algorithms are executed under the same environmental conditions and robot constraints. JIST and OBL plan in the space of position and velocity of the robot,

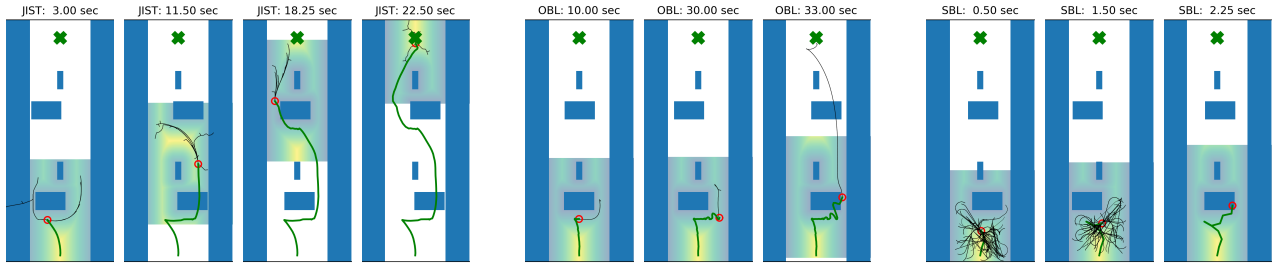


Fig. 4: Same Patrol benchmark example (left to right) where JIST (ours) (left) successfully reaches the goal while OBL (middle) and SBL (right) terminate in collision.

while SBL only plans in space of robot position to which average velocity is applied for execution.

Environments In our benchmark environments, **2D Static**, **2D Forest**, **Patrol**, **Pedestrian**, and **3D Forest**, we assume that the robot is only aware of the locations of obstacles in a local neighborhood (that corresponds to real world LIDAR ranges) and not their velocities. We also incorporate stochastic actions and noisy state measurements for the robots (with Gaussian noise). In all our experiments, we individually tuned all the hyperparameters in each algorithm to get their best performance.

Metrics We record the following metrics that are averaged over a set of trials, run with unique random seeds shared by all algorithms such that they encounter the exact same set of planning problems (start, goal, and environment): (i) rate of success of reaching goal without getting into collisions or timing out (**Success**), (ii) total execution i.e. wall-clock time taken by the robot to reach the goal from the start (**Execution time**) averaged over successful runs, (iii) average computation time per iteration taken by the algorithm to plan (**Compute time**) measured starting from the input from the perception module (e.g. signed distance field of the environment) to the output plan from the algorithm, and (iv) total distance traveled by the robot normalized by the starting euclidean distance to goal (**Norm. Dist**) averaged over successful runs.

A. 2D Static: Static Benchmark for Navigation

We begin with a benchmark in a static setting to establish the starting performance of our baselines and show that they excel here, as discussed in Section I, as well as demonstrate that our method performs competitively in this setting. We consider the task of planar point goal (x, y, yaw) navigation in a large environment with even distribution of uniformly spaced static square shaped obstacles on a grid. In this environment, we consider a differential drive robot (non-holonomic constraint) that is handled in JIST and OBL with a unary factor on any state while for SBL we use Reeds-Shepp motion curves [49] when extending and connecting states on the tree. We also enforce velocity limits on the robot for all algorithms (as factors for the former two). 30 random trials are generated where the start and goal are at least a fixed minimum distance apart.

The benchmark results are summarized in Table I (row 1). We observe that all methods are successfully able to reach the goal in every run and the difference in the other metrics

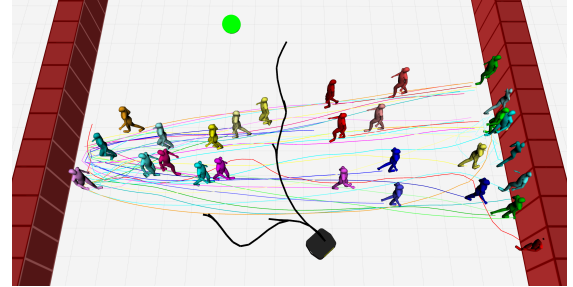


Fig. 5: Pedestrian benchmark example in Gazebo with JIST (ours).

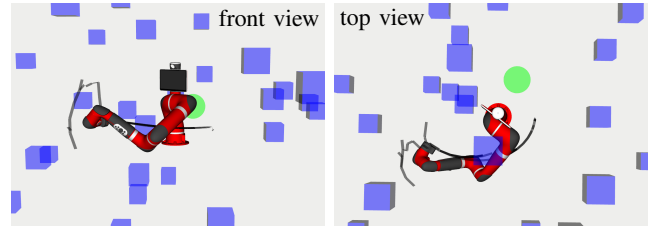


Fig. 6: 3D Forest benchmark example with JIST (ours).

reveal the underlying key algorithmic differences. Sampling enabled exploration allows JIST and SBL to reach the goal more quickly (lower execution time). The optimization property leads to lower compute times for JIST and OBL along with smoother trajectories and lower normalized distances compared to SBL. JIST naturally incurs a slightly higher compute time since it is optimizing a more complex tree shaped factor graph with more connection compared to a single chain like factor graph of OBL.

B. 2D Forest: Random Forest Benchmark for Navigation

The primary benchmark task we consider is planar point goal (x, y, yaw) navigation in a highly dynamic 2D forest with large number of randomly moving square shaped obstacles that can accelerate in random directions as shown in Fig. 3. We use the same differential drive robot and the same costs and factors for the algorithms from the 2D Static benchmark. The robot's limited visibility is also shown by the shaded square in Fig. 3 that represents the instantaneous signed distance field (SDF) used for collision checking.

The benchmark results are summarized in Table I (row 2). Under the increased difficulty of the problem compared to the static setting, we observe drop in performance for all algorithms. But, JIST is significantly more successful at reaching the goal compared to both the baselines and reaching the goal faster compared to OBL with nearly the same amount of computation time differences as in the

TABLE I: Benchmark results on all environments.

	Success			Execution time (s)			Compute time (s)			Norm. Dist		
	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL
2D Static	1.00	1.00	1.00	28.08	51.53	34.67	0.016	0.011	0.252	1.38	1.41	1.65
2D Forest	0.43	0.20	0.03	47.75	75.50	17.75	0.016	0.011	0.325	1.95	1.98	1.88
Patrol	0.60	0.27	0.07	18.32	27.03	7.75	0.166	0.385	0.451	1.26	1.18	2.07
Pedestrian	0.70	0.45	0.00	288.04	170.53	-	0.039	0.035	-	4.94	2.15	-
3D Forest	0.71	0.60	0.36	18.64	18.69	11.83	0.140	0.256	0.403	5.37	3.29	6.51

TABLE II: Ablations on 2D Forest environment.

Environment obstacle number ablation												
	Success			Execution time (s)			Compute time (s)			Norm. Dist		
	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL
20	0.97	0.90	0.37	21.45	43.94	18.98	0.014	0.011	0.325	1.11	1.25	1.54
40	0.80	0.67	0.13	28.27	59.38	19.12	0.015	0.011	0.304	1.32	1.58	1.78
60	0.57	0.63	0.13	34.72	66.82	19.44	0.016	0.011	0.326	1.54	1.75	1.77
80	0.43	0.20	0.03	47.75	75.50	17.75	0.015	0.011	0.320	1.95	1.98	1.88
100	0.40	0.07	0.03	47.58	92.50	23.75	0.015	0.011	0.371	1.85	2.48	1.77
Environment obstacle top speed ablation (m/s)												
	Success			Execution time (s)			Compute time (s)			Norm. Dist		
	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL
0.5	0.63	0.53	0.30	78.61	135.83	24.69	0.015	0.011	0.388	2.46	2.60	2.17
1.0	0.53	0.40	0.10	51.78	87.56	19.83	0.017	0.011	0.305	1.92	2.03	1.76
1.5	0.43	0.20	0.03	47.75	75.50	17.75	0.016	0.011	0.325	1.95	1.98	1.88
2.0	0.30	0.10	0.00	45.22	63.67	-	0.016	0.011	-	1.86	1.80	-
2.5	0.40	0.17	0.03	40.29	56.90	20.50	0.016	0.011	0.329	1.83	1.74	1.74
Execution noise ablation (m)												
	Success			Execution time (s)			Compute time (s)			Norm. Dist		
	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL
0.0	0.43	0.27	0.00	43.06	66.75	-	0.015	0.010	-	1.67	1.77	-
0.05	0.33	0.23	0.10	40.77	70.46	16.67	0.015	0.010	0.268	1.61	1.95	1.72
0.1	0.40	0.23	0.13	43.38	82.68	22.44	0.015	0.010	0.329	1.72	1.95	1.87
0.15	0.60	0.17	0.10	51.15	166.55	19.42	0.015	0.010	0.260	1.99	2.15	1.81
0.2	0.50	0.17	0.07	46.60	137.50	21.12	0.015	0.011	0.263	1.82	2.46	2.08
Node budget ablation												
	Success			Execution time (s)			Compute time (s)			Norm. Dist		
	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL	JIST	OBL	SBL
40	0.43	0.30	0.07	45.87	77.83	20.12	0.010	0.007	0.188	1.69	1.97	1.53
60	0.43	0.20	0.03	47.75	75.50	17.75	0.016	0.011	0.324	1.95	1.98	1.88
80	0.37	0.27	0.03	39.93	61.53	21.00	0.020	0.013	0.429	1.75	1.68	2.14
100	0.53	0.27	0.00	41.42	59.00	-	0.027	0.017	-	1.83	1.70	-
120	0.43	0.23	0.07	33.79	74.14	19.20	0.034	0.021	0.496	1.74	2.10	1.89

2D Static benchmark. This shows that the JIST is able to leverage the exploration enabled by sampling but isn't burdened by a static graph that is more expensive to update. Instead it can exploit optimization to quickly reason over the multiple potential plans. On the other hand, SBL retains a static tree and only removes nodes or edges in collision, but cannot move them a safe distance away from obstacles as done during optimization. OBL cannot explore and is only able to optimize and reason over only one potential plan.

To study how the performance gap between the three algorithms varies as the problem difficulty scales, we also carry out ablation studies in this environment and are presented in Table II. First, we vary the number of dynamic obstacles and their speeds to scale the difficulty. We observe that starting at the lowest difficulty that is closest to the static setting SBL already drops in success. As the problem becomes more difficult the success rate of the baselines falls close

to zero, while JIST only gradually falls to 0.4. Increasing difficulty also increases the execution time and normalized distance, but all algorithms maintain similar differences in their compute times. The outliers are due to the small number of successful outcomes in those settings. Then, we vary the variance of noise in measuring the current state of the robot and the noise in executing actions. All algorithms are able to maintain their success rate given the benefits of replanning, but incur slightly higher execution times and normalized distance under high noise. Finally, we vary the available node budget that corresponds to the cap on the maximum number of nodes in the tree for JIST and SBL, and the length of horizon for OBL. We observe that the computation times increases with increased node budget with minor improvements in success rate. Later in Section V we discuss alternate ways of sampling that can be explored to improve performance.

C. Patrol: Dynamic Narrow Passage Benchmark

Next, we benchmark all three algorithms in the setting of robot navigation through a narrow passages guarded by dynamic obstacles as shown in Fig. 4 and is a more complex version of the patrol guard environment from [46]. Here the passage consists of two rectangular obstacles moving back and forth along a horizontal line with random velocity at two different locations in the passage. For this environment, we use the same robot agent, algorithm specific considerations, and noise models as we did in the 2D Forest benchmarks. In these experiments we randomly sample a start and goal at both ends of the passage respectively. Results averaged over 30 trials are summarized in Table I (row 3). Our findings are in the vein of the previous benchmarks where we observe JIST is able to outperform in success rate over the baseline while incurring a small added computation time over OBL which is still overall much lower compared to SBL.

D. Pedestrian: Benchmark in Gazebo Simulator

We also perform a benchmark in a more practical navigation setting where the robot has to reach a goal while avoiding pedestrians as shown in Fig. 5. In this setting, we make the experiments realistic and real-time by deploying the entire system in the Gazebo [50] physics simulator. We simulate the robot's perception to build SDFs from LIDAR scans and simulate the robot's control for execution using PID velocity controllers. We use the ClearPath Ridgeback mobile robot that has an omni-directional drive and do not use the holonomic constraint, but still enforce velocity limits. We also simulate realistic pedestrian motion in crowds using Pedsim [51]. The results from this benchmark are averaged over 100 trials and presented in Table I (row 4) and shows JIST similarly outperforms both baselines.

E. 3D Forest: Random Forest Benchmark for Manipulation

Lastly, we benchmark a manipulation task as shown in Fig. 6. Here, we consider online planning for a seven degree of freedom Sawyer robot arm in a dynamic environment with cubes moving with random velocities. Robot joint limit and self-collision avoidance is enforced for all algorithms (as factors for JIST and OBL). The robot has full visibility of its workspace since the manipulator base remains fixed. The goal for the robot is specified as a target end-effector location. For the experiments we randomly sample a start joint configuration for the robot arm and a reachable task space goal location. The normalized distance metric is computed using the task space distance traveled by the end-effector. We present the averaged results over 100 trials in Table I (row 5) and find results consistent with the other benchmarks.

V. DISCUSSION

In this work, we focused on the problem of motion planning in large, dense, highly dynamic environments and identified the essence of the problem as the ability to efficiently grow, track, and switch between multiple possible candidate solutions. We validated this with our approach JIST that combines the exploration and exploitation strengths of sampling and optimization respectively. In this approach

we do not check the samples or the edge connections made during sampling for collision and instead let the optimizer try to move the samples and the edges out of collision. In doing so we gain practical efficiency but sacrifice the theoretical properties like probabilistic completeness offered by sampling methods.

We currently precompute the SDFs and cache them for the 3D environments. While SDFs can be computed online for 2D environments, in 3D settings, solutions to building SDFs incrementally online on GPUs [52] can be utilized to support the perception input for our approach and maintain real time requirements for the overall system.

While in this specific implementation we used random sampling and nonlinear least square optimization, our framework is flexible enough to support alternate sampling and optimization strategies with the underlying factor graph acting as the core data structure tying them together. To mitigate local minima or bias sampling toward other desired outcomes it is possible to use goal directed sampling [53], learn heuristics [54] or learn to sample [55]. Similarly, optimization can be adapted to support learned cost functions and weights [56].

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] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1, pp. 93 – 146, 2004.
- [3] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in neural information processing systems*, 2004, pp. 767–774.
- [4] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm." in *ICAPS*, 2005, pp. 262–271.
- [5] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 1243–1248.
- [6] 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.
- [7] A. Mandalika, O. Salzman, and S. Srinivasa, "Lazy receding horizon A* for efficient path planning in graphs with expensive-to-evaluate edges," *arXiv preprint arXiv:1803.04998*, 2018.
- [8] 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.
- [9] 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.
- [10] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
- [11] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.
- [12] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.
- [13] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII*. Springer, 2009, pp. 449–464.
- [14] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Robotics Research*. Springer, 2003, pp. 403–417.

- [15] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1478–1483.
- [16] 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.
- [17] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.
- [18] 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.
- [19] M. Toussaint, "Robot trajectory optimization using approximate inference," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 1049–1056.
- [20] 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.
- [21] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4569–4574.
- [22] 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.
- [23] 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.
- [24] H. Lee, D. Lee, and D. H. Shim, "Receding horizon-based RRT* algorithm for a UAV real-time path planner," in *AIAA Information Systems-AIAA Infotech@ Aerospace*, 2017, p. 0676.
- [25] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 1603–1609.
- [26] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How, "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns," *Autonomous Robots*, 2013.
- [27] M. W. Otte and E. Frazzoli, "RRTX: Real-time motion planning/replanning for environments with unpredictable obstacles," in *WAFR*, 2014.
- [28] 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)*. IEEE, 2017, pp. 1416–1421.
- [29] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming," in *Advances in neural information processing systems*, 2008, pp. 1465–1472.
- [30] F. Allgöwer and A. Zheng, *Nonlinear model predictive control*. Birkhäuser, 2012, vol. 26.
- [31] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments," in *ICAPS*, 2012.
- [32] 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.
- [33] C. Park, J. Pan, and D. Manocha, "Real-time optimization-based planning in dynamic environments using GPUs," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 4090–4097.
- [34] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 5332–5339.
- [35] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.
- [36] S. Quinlan, "Real-time modification of collision-free paths," Ph.D. dissertation, Stanford University, 1994.
- [37] J.-H. Chuang and N. Ahuja, "An analytically tractable potential field model of free space and its application in obstacle avoidance," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 28, no. 5, pp. 729–736, 1998.
- [38] M. Mabrouk and C. McInnes, "Solving the potential field local minimum problem using internal agent states," *Robotics and Autonomous Systems*, vol. 56, no. 12, pp. 1050–1060, 2008.
- [39] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*, 2008.
- [40] A. Kuntz, C. Bowen, and R. Alterovitz, "Interleaving optimization with sampling-based motion planning (ios-mp): Combining local optimization with global exploration," *arXiv preprint arXiv:1607.06374*, 2016.
- [41] B. S. Willey, "Combining sampling and optimizing for robotic path planning," Ph.D. dissertation, Rice University, 2018.
- [42] 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.
- [43] 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.
- [44] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 498–519, 2001.
- [45] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.
- [46] K. Kolar, S. Chintalapudi, B. Boots, and M. Mukadam, "Online motion planning over multiple homotopy classes with Gaussian process inference," *IEEE Conference on Robotics and Automation (ICRA)*, 2019.
- [47] F. Dellaert, "Factor graphs and GTSAM: a hands-on introduction," Georgia Tech Technical Report, GT-RIM-CP&R-2012-002, Tech. Rep., 2012.
- [48] M. Mukadam, C.-A. Cheng, X. Yan, and B. Boots, "Approximately optimal continuous-time motion planning and control via probabilistic inference," in *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.
- [49] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.
- [50] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.
- [51] C. Gloor, "Pedsim: Pedestrian crowd simulation," URL <http://pedsim.silmaril.org>, vol. 5, no. 1, 2016.
- [52] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (iros)*. IEEE, 2017, pp. 1366–1373.
- [53] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2997–3004.
- [54] S. Choudhury, M. Bhardwaj, S. Arora, A. Kapoor, G. Ranade, S. Scherer, and D. Dey, "Data-driven planning via imitation learning," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1632–1672, 2018.
- [55] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.
- [56] M. Bhardwaj, B. Boots, and M. Mukadam, "Differentiable Gaussian process motion planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10598–10604.
- [57] T. Barfoot, C. H. Tong, and S. Sarkka, "Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression," *Proceedings of Robotics: Science and Systems, Berkeley, USA*, 2014.

A. Factor Descriptions

In this section we provide details of all the factors used in the JIST factor graph in our current experiments.

GP prior factor ensure smoothness in the resulting trajectory and encodes the optimality criteria in the objective function [23]. It is derived from a linear time varying system [57] meant to represent the motion model of the robot.

$$\mathbf{f}^{gp}(\theta_t^i, \theta_{t+1}^j) = \exp \left\{ -\frac{1}{2} \|\Phi_{t,t+1} \theta_t^i - \theta_{t+1}^j\|_{\mathbf{Q}_{t,t+1}}^2 \right\}$$

$$\Phi_{t,t+1} = \begin{bmatrix} \mathbf{I} & \Delta t \mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \quad \mathbf{Q}_{t,t+1} = \begin{bmatrix} \frac{1}{3} \Delta t^3 \mathbf{Q}_c & \frac{1}{2} \Delta t^2 \mathbf{Q}_c \\ \frac{1}{2} \Delta t^2 \mathbf{Q}_c & \Delta t \mathbf{Q}_c \end{bmatrix}$$

where $\Phi_{t,t+1}$ is the state transition matrix during time interval Δt between t and $t+1$, with variance $\mathbf{Q}_{t,t+1}$ and hyperparameter \mathbf{Q}_c .

Obstacle avoidance factor on any state is used to keep that state collision free from obstacles in the environment.

$$\mathbf{f}^{obs}(\theta_t^i) = \exp \left\{ -\frac{1}{2} \|\mathbf{c}(\mathbf{d}(\phi(\theta_t^i)))\|_{\Sigma^{obs}}^2 \right\}$$

where $\Sigma^{obs} = \sigma_{obs}^2 \mathbf{I}$ and ϕ maps the robot configuration to a set of representative points on the robot body in task space that are queried in the SDF \mathbf{d} to get their signed distances. Hinge loss cost function \mathbf{c} with safety distance ϵ then gives the obstacle collision cost given the signed distances [23].

Binary obstacle avoidance factor between two consecutive states uses fast GP interpolation [57] \mathbf{p} to find the state at time τ in the interval $(t, t+1)$ and the same hinge loss cost for collision avoidance. A set of these together in between states ensures that the edge is also evaluated for collision [23].

$$\mathbf{f}^{intp}(\theta_t^i, \theta_{t+1}^j; \tau) = \exp \left\{ -\frac{1}{2} \|\mathbf{c}(\mathbf{d}(\phi(\mathbf{p}(\theta_t^i, \theta_{t+1}^j, \tau))))\|_{\Sigma^{obs}}^2 \right\}$$

Start or current state factor is used to ground the trajectory at the start or current state and also represents the measurement θ_{curr} of the current state with variance $\Sigma^{curr} = \sigma_{curr}^2 \mathbf{I}$.

$$\mathbf{f}^{curr}(\theta_t^i) = \exp \left\{ -\frac{1}{2} \|\theta_t^i - \theta_{curr}\|_{\Sigma^{curr}}^2 \right\}$$

Goal factor is used on every state except the current state to drive all path towards the goal. The strength of the factor is based on the proximity of the current state to the goal [48].

$$\mathbf{f}^{goal}(\theta_t^i) = \exp \left\{ -\frac{1}{2} \|\theta_t^i - \theta_{goal}\|_{\Sigma^{goal}}^2 \right\}$$

$$\Sigma^{goal} = \sigma_{goal}^2 \frac{\|\theta_{curr} - \theta_{goal}\|^2}{\|\theta_{start} - \theta_{goal}\|^2} \mathbf{I}$$

Joint and velocity limit factor are enforced with a similar hinge loss cost function like obstacle avoidance where a penalty is incurred if the state gets ϵ close or over the limit

in any dimension of the state.

$$\mathbf{f}^{lim}(\theta_t^i) = \exp \left\{ -\frac{1}{2} \|\mathbf{c}(\theta_t^i)\|_{\Sigma^{lim}}^2 \right\}$$

$$c = \begin{cases} ll + \epsilon - \theta_t^i & \text{if } \theta_t^i < ll + \epsilon \\ \theta_t^i - ul - \epsilon & \text{if } ul + \epsilon < \theta_t^i \\ 0 & \text{if } ll + \epsilon \leq \theta_t^i \leq ul + \epsilon \end{cases}$$

where c evaluates the cost for each scalar dimension θ_t^i of state vector θ_t^i and \mathbf{c} stacks all c together. ul and ll are the upper and lower limits respectively and $\Sigma^{lim} = \sigma_{lim}^2 \mathbf{I}$.

Non-holonomic constraint factor is used for planar robots with state $\theta_t^i = [x_t^i, y_t^i, \psi_t^i, \dot{x}_t^i, \dot{y}_t^i, \dot{\psi}_t^i]^\top$ with a differential drive system to restrict implausible sideways motion of the robot.

$$\mathbf{f}^{nh}(\theta_t^i) = \exp \left\{ -\frac{1}{2} \|\dot{y}_t^i \cos(\psi_t^i) - \dot{x}_t^i \sin(\psi_t^i)\|_{\sigma_{nh}^2}^2 \right\}$$

Self collision avoidance factor is useful for manipulators with high degree of freedom to avoid self collision. We use a similar hinge loss cost function like the obstacle avoidance except here the distance function \mathbf{d} calculates the distances of all representative points on the robot body with respect to each other. The pairs of points to be checked are pre-specified and optimized to not check pairs of points for instance that are on the same link of the robot.

$$\mathbf{f}^{self}(\theta_t^i) = \exp \left\{ -\frac{1}{2} \|\mathbf{c}(\mathbf{d}(\phi(\theta_t^i)))\|_{\Sigma^{self}}^2 \right\}$$

where $\Sigma^{self} = \sigma_{self}^2 \mathbf{I}$.

B. Experimental Details

In this section we provide additional details on the experiments.

1) *Environments and robots*: In all navigation experiments the top allowed speed for the robots is 3m/s for translation and 0.6rad/s for rotation. **2D Static** and **2D Forest** environments are 90m \times 120m in size. All obstacles are square shaped with side length 6m and move in the environment with random acceleration limited between -0.6m/s² and 0.6m/s² in random directions (zero velocity and acceleration in static case). 2D Static environment comprised of 48 obstacles uniformly spaced at 15m on a grid and the 2D Forest consisted of 80 randomly distributed dynamic obstacles. The robot is circular with a radius of 1.5m. We considered a Gaussian noise for both the robot dynamics model and robot localization model with a mean of [0.03m, 0.03m, 0.03 radians]. **Patrol** environment is 10m \times 40m in size with two moving and two stationary rectangular obstacles. Robot is the same as in 2D Forest but with a smaller radius of 0.5m. **Pedestrian** environment is 14m \times 15m in size. The pedestrians have a top speed of 2m/s and follow a swarm-like crowd motion, simulated using [51]. At any given point, the number of pedestrians on the map range from 15 to 40 in our experiments and is not fixed as the simulator lets pedestrians enter and leave in a continuous stream. **3D Forest** environment consists of 50 randomly moving cubical obstacles of size length 0.15m with top speed of 0.5m/s. The obstacles are prevented from colliding

with the base link of the manipulator as it is fixed in the environment. The top allowed speed for all joints of the manipulator is 1rad/s

2) *Algorithms*: Hyperparameters of all factors and cost functions are tuned individually for each algorithm. Constraints are handled with factors for JIST and OBL, while for SBL we perform explicit collision checking, velocity limit check, robot dynamics and kinematic constraint validation while adding new states and edges to the tree. To produce dynamically efficient paths that are also at a safe distance away from the obstacles, in SBL, we use the same cost function terms that we used to model the GP factor and obstacle avoidance factor. We also bias the search towards the goal for SBL by simply augmenting the cost function with a goal heuristic (weighted euclidean distance to the goal) from any given state on the tree. Unlike the configuration position and velocity based state in JIST and OBL, the state in SBL only consists of configuration position. Thus, SBL provides a path that we then post process with linear interpolation to fit a trajectory. In navigation experiments, while growing the trees in JIST and SBL, we limit the sampling to a local neighborhood around the robot. This encourages the algorithms to explore more in the local visible neighborhood of the robot which has a higher influence on its immediate future.