2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

PINSAT: Parallelized Interleaving of Graph Search and Trajectory Optimization for Kinodynamic Motion Planning

Ramkumar Natarajan[1], Shohin Mukherjee[1], Howie Choset[1] and Maxim Likhachev[1]

***Abstract*— Trajectory optimization is a widely used technique in robot motion planning for letting the dynamics of the system shape and synthesize complex behaviors. Several previous works have shown its benefits in high-dimensional continuous state spaces and under differential constraints. However, long time horizons and planning around obstacles in non-convex spaces pose challenges in guaranteeing convergence or finding optimal solutions. As a result, discrete graph search planners and sampling-based planers are preferred when facing obstacle-cluttered environments. A recently developed algorithm called INSAT effectively combines graph search in the low-dimensional subspace and trajectory optimization in the full-dimensional space for global kinodynamic planning over long horizons. Although INSAT successfully reasoned about and solved complex planning problems, the numerous expensive calls to an optimizer resulted in large planning times, thereby limiting its practical use. Inspired by the recent work on edge-based parallel graph search, we present PINSAT, which introduces systematic parallelization in INSAT to achieve lower planning times and higher success rates, while maintaining significantly lower costs over relevant baselines. We demonstrate PINSAT by evaluating it on 6 DoF kinodynamic manipulation planning with obstacles. We demonstrate PINSAT by evaluating it on two kinodynamic manipulation planning scenarios: (i) a single ball blocking task among obstacles using a 6 DoF ABB arm, and (ii) a multi-ball blocking task where the balls are separated by short time intervals using a 7 DoF KUKA LBR iiwa arm with obstacles.**

I. Introduction

Graph search-based planning algorithms like A* and its variants [1], [2], [3] enable robots to come up with well-reasoned long-horizon plans to achieve a given task objective [4], [5]. They do so by searching over the graph that results from discretizing the state and action space. However, in robotics, several dynamically rich tasks require high-dimensional planning in the continuous space. For such domains, kinodynamic planning and trajectory optimization techniques have been developed to synthesize dynamically feasible trajectories. The existing kinodynamic algorithms achieve this by discretizing the action space to roll out trajectory primitives in the discrete or continuous state space. On the other hand, trajectory optimization techniques do not discretize the state or action space but suffer local minima, lack convergence guarantees in nonlinear settings, and struggle to reason over long horizons.

An algorithm called INSAT [6], [7], [8], short for INterleaved Search And Trajectory optimization, bridges this gap for global kinodynamic planning. The idea behind INSAT was (a) to identify a low-dimensional manifold, (b) perform a search over a discrete graph embedded in this manifold, and (c) while searching the graph, utilize full-dimensional trajectory optimization to compute the cost of partial solutions found by the search. Thus every edge/action evaluation in the graph required solving at least one and potentially several instances of trajectory optimization. Though INSAT used dynamic programming to cleverly warm-start every instance of optimization with a good approximate solution, the frequent calls to the optimizer limited its practical usage.

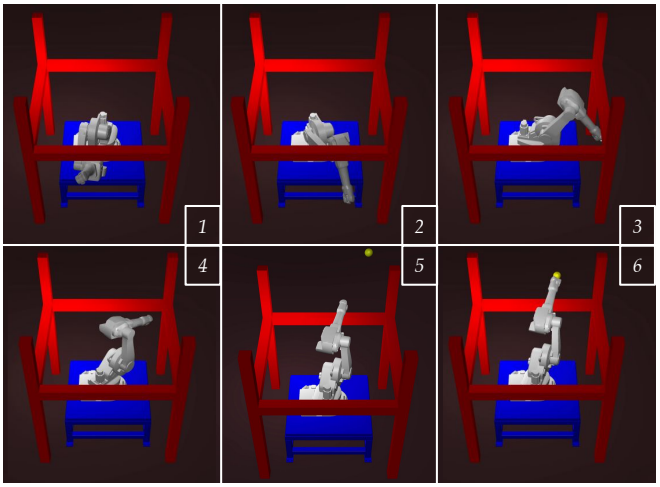

Fig. 1: ABB arm evading obstacles to block a ball thrown at it using kinodynamic motion produced by PINSAT.

For domains where action evaluation is expensive, a parallelized planning algorithm ePA*SE (Edge-based Parallel A* for Slow Evaluations) was developed [9] that changes the basic unit of the search from state expansions to edge expansions. This decouples the evaluation of edges from the expansion of their common parent state, giving the search the flexibility to figure out which edges need to be evaluated to solve the planning problem. In this work, we employ the parallelization technique of ePA*SE to parallelize the expensive trajectory optimization step in INSAT. The resulting algorithm termed PINSAT: Parallel Interleaving of Graph Search and Trajectory Optimization, can compute dynamically feasible plans while achieving close to real-time performance. The key motivation for developing PINSAT is that if INSAT can effectively solve long-horizon, dynamically rich planning tasks, then using the ideas of ePA*SE parallelization to expedite slow edge optimizations in INSAT will result in a strictly better algorithm. We empirically demonstrate that this is indeed the case in PINSAT.

II. Related Work

1) Kinodynamic Motion Planning: Kinodynamic planning has been a subject of extensive research, driven by the need for robots and autonomous systems to navigate complex environments while considering their dynamic constraints. This dynamic feasibility is typically achieved using four different schemes namely (i) search-based, (ii) sampling-based, (iii) optimization-based, and (iv) hybrid methods.

[1] The authors are with The Robotics Institute at Carnegie Mellon University, Pittsburgh PA 15213. email: {rnataraj, shohinm, maxim, choset}@cs.cmu.edu

Search-based kinodynamic planning involves precomputing a set of short, dynamically feasible trajectories called motion primitives that capture the robot's capabilities. Search algorithms like A* or its variants [10] are used to search over a graph wherein for any state its successor states are being computed by applying motion primitives and provide guarantees of optimality and completeness w.r.t. the chosen primitives. However, the choice and calculation of these motion primitives that are efficient can be challenging, particularly in high-dimensional systems. **Sampling-based** kinodynamic methods adapt and extend classic approaches such as Probabilistic Roadmaps (PRM) [11] and Rapidly Exploring Random Trees (RRT) [12] to handle dynamic systems [13]. This is achieved using dynamically feasible rollouts with random control inputs or solutions of boundary value problems within the *extend* operation. There are probabilistically complete and asymptotically optimal variants [14], [15], however, empirical convergence might be tricky. **Optimization-based** planning methods can generate high-quality trajectories that are dynamically feasible and do not suffer from the curse of dimensionality in search-based methods. They formulate the motion planning problem as an optimization problem [16] with cost functions defined for trajectory length, time, or energy consumption. After transcribing into a finite-dimensional optimization, these methods rely on the gradients of the cost function and dynamics and employ numerical optimization algorithms to find locally optimal solutions [17], [18], [19]. However, except for a small subset of systems (such as linear or flat systems), for most nonlinear systems these methods lack guarantees on completeness, optimality or convergence. **Hybrid** planning methods combine two or all of the schemes mentioned above. Search and sampling methods are combined in [20], [21], [22], sampling and optimization methods are combined in [23], [24], [25], search, sampling, and optimization methods are combined in [26]. INSAT combined search and optimization methods and demonstrated its capability in several complex dynamical systems [6], [7].

2) Parallel Search: Several approaches parallelize sampling-based planning algorithms in which parallel processes cooperatively build a PRM [27] or an RRT [28], [29], [30] by sampling states in parallel. However, in many planning domains, sampling of states is not trivial. One such class of planning domains is simulator-in-the-loop planning, which uses a physics simulator to generate successors [31]. Unless the state space is simple, such that the sampling distribution can be scripted, there is no principled way to sample meaningful states that can be realized in simulation.

Search-based methods like A* can be parallelized by generating successors concurrently during state expansion, but are limited by the domain's branching factor. Alternatively, PA*SE [32] overcomes parallelization limited by branching factor and expands states at most once without affecting bounds on solution quality. However, PA*SE is inefficient in domains with costly edge evaluations, as each thread sequentially evaluates the outgoing edges. To address this, ePA*SE [9] improves PA*SE by parallelizing edge search. MPLP [33] achieves faster planning by lazily running the search and asynchronously evaluating edges in parallel, assuming that successors can be generated without evaluating the edge. Some work focuses on parallelizing A* on GPUs [34], [35], but their SIMD execution model limits them to domains with simple actions sharing the same code.

To the best of our knowledge, there is very little previous work on parallel kinodynamic planning. All existing approaches are sampling-based and employ a straightforward parallelization to execute the *steer* operation in batch—either on CPU for steering with boundary value solvers [36] or on the GPU for steering with neural networks in batch [37]. Consequently, they inherit the same limitations as sampling-based planners, where the sampling of states can be nontrivial. Search-based methods overcome this limitation by systematically exploring the space, and INSAT & PINSAT owe their success to this characteristic.

III. Problem Formulation

Let the n-dimensional state space of the robot be denoted by $\mathcal{X} \subseteq \mathbb{R}^n$. Let $\mathcal{X}^{obs} \subset \mathcal{X}$ be the obstacle space and $\mathcal{X}^{\text{free}} = \mathcal{X} \setminus \mathcal{X}^{obs}$ be the free motion planning space. For kinodynamic motion planning, we should reason about and satisfy constraints on the derivatives of position such as velocity, acceleration, etc. This boils down to including those derivatives as a part of the planning state and can quickly lead to a large intractable planning space. To cope with this let us consider a low-dimensional (low-D) space $\mathcal{X}_L$ and an auxiliary space $\mathcal{X}_A$ whose Cartesian product forms the full-dimensional (full-D) space $\mathcal{X} = \mathcal{X}_A \times \mathcal{X}_L$. Let the low-D space be such that $\mathcal{X}^{obs} \subset \mathcal{X}_L$. Consider a finite graph $G = (\mathcal{V}, \mathcal{E})$ defined as a set of vertices $\mathcal{V}$ and directed edges $\mathcal{E}$ embedded in this low-D space $\mathcal{X}_L$. Each vertex $v \in \mathcal{V}$ represents a state $\mathbf{x} \in \mathcal{X}_L$. An edge $e \in \mathcal{E}$ connecting two vertices v_1 and v_2 in the graph represents an action $\mathbf{a} \in \mathcal{A}$ that takes the agent from corresponding states $\mathbf{x}_1$ to $\mathbf{x}_2$. In this work, we assume that all actions are deterministic. Hence an edge e can be represented as a pair $(\mathbf{x}, \mathbf{a})$, where $\mathbf{x}$ is the state at which action $\mathbf{a}$ is executed. For an edge e, we will refer to the corresponding state and action as $e.\mathbf{x}$ and $e.\mathbf{a}$ respectively. So given a start state $\mathbf{x}^S$ and a goal region $\mathcal{G}$, the kinodynamic motion planning problem can be cast as the following optimization problem

$$\min_{\boldsymbol{\phi}(t), T} \quad w_1 T + w_2 \int_0^T \boldsymbol{\phi}(t) dt \tag{1a}$$

$$\text{s.t.} \quad 0 < t_{\min} \leq T \leq t_{\max} \tag{1b}$$

$$\boldsymbol{\phi}(0) = \mathbf{x}^S, \boldsymbol{\phi}(T) = \mathbf{x}^G \in \mathcal{G} \tag{1c}$$

$$|\boldsymbol{\phi}^{(j)}(t)| \preceq \boldsymbol{\phi}^{(j)}_{\text{lim}} > 0; \forall t; j = \{1, 2, \ldots, \gamma\} \tag{1d}$$

$$\boldsymbol{\phi}(t) \in \mathcal{X}^{\text{free}}; \forall t \tag{1e}$$

$$\boldsymbol{\phi}(t) \in \mathcal{C}^{\gamma}(\mathbb{R}); \forall t \tag{1f}$$

where $\boldsymbol{\phi} : [0, T] \to \mathcal{X}_L$ is a spatial trajectory that is γ-times differentiable and therefore continuous, w_1 and w_2 trade-off the relative importance between path length and duration, and $\boldsymbol{\phi}^{(j)}(t) = \frac{d^j \boldsymbol{\phi}(t)}{dt^j}$ is the j-th derivative of $\boldsymbol{\phi}(t)$. The Eq. 1b is the constraint on the maximum duration of the trajectory, Eq. 1c is the boundary start and goal conditions, Eq. 1d is the element-wise system limit on the derivatives of the positional trajectory, Eq. 1e requires the trajectory to lie in $\mathcal{X}^{\text{free}}$ and Eq. 1f imposes the degree of continuity on the trajectory. The minimizer of the Eq. 1 is a trajectory $\boldsymbol{\phi}_{\mathbf{x}^S \mathbf{x}^G}(t)$[1] that connects start state $\mathbf{x}^S$ to the goal region $\mathcal{G}$. There is a computational budget of N_t threads available, which can run in parallel.

[1]The argument t of the $\boldsymbol{\phi}(t)$ may be dropped for brevity.

IV. Background

In kinodynamic planning, a great deal of focus is on the integration accuracy of the equations of motion to obtain feasible solutions to the dynamic constraints. However, fully controllable systems, like the vast majority of commercial manipulators, can accurately track trajectories generated with smooth splines [38] even if they are mildly inconsistent with the system's nonlinear dynamics. Even for many other dynamical systems, parameterizing trajectories with polynomials achieves near-dynamic feasibility and proves to be highly effective. Popular techniques like direct collocation and pseudo-spectral methods parameterize trajectories with polynomials at some level. In this work, we use a piecewise polynomial widely used recently for motion planning [39], [40] called B-spline to represent trajectories. We will first provide some background on B-splines and highlight a few nice properties that enable us to satisfy dynamics and guarantee completeness.

A. B-Splines

B-splines are smooth and continuous piecewise polynomial functions made of finitely many basis polynomials called B-spline bases. A k-th degree B-spline basis with m control points can be calculated using the Cox-de Boor recursion formula [41] as

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \tag{2}$$

where $i = 0, \ldots, m$, $\frac{t-t_i}{t_{i+k-1}-t_i}$ and $\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}}$ are the interpolating coefficients between t_i and t_{i+k}. For $k = 0$, $N_{i,0} = 1$ if $t_i \leq t < t_{i+1}$ and 0 otherwise. Let us define a non-decreasing knot vector $\mathbf{T}$ and the set of control points $\mathbf{P}$ called de Boor points

$$\mathbf{T} = \{\underbrace{t_0, \ldots, t_0}_{k-\text{points}}, t_{k+1}, \ldots, t_m, \underbrace{t_f, \ldots, t_f}_{k-\text{points}}\} \tag{3a}$$

$$\mathbf{P} = \{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_m\} \tag{3b}$$

where $\mathbf{p}_i \in \mathbb{R}^n, i = 0, \ldots, m$. Then our B-spline trajectory $\boldsymbol{\phi}(t)$ can be uniquely determined by $\mathbf{T}$, $\mathbf{P}$ and the degree of the polynomial k

$$\boldsymbol{\phi}(t) = \sum_{i=0}^{m} \mathbf{p}_i N_{i,k}(t) \tag{4}$$

Remark 1. The above B-spline trajectory $\boldsymbol{\phi}(t)$ is guaranteed to entirely lie inside the convex hull of the active control points $\mathbf{P}$ (the control points whose bases are not zero).

We need to evaluate the derivatives of $\boldsymbol{\phi}(t)$ to satisfy the limits on joint velocity and acceleration. For this, the j-th derivative of $N_{i,k}(t)$ is given as

$$N^{(j)}_{i,k}(t) = j \left(\frac{N^{(j-1)}_{i,k-1}(t)}{t_{i+k} - t_i} - \frac{N^{(j-1)}_{i+1,k-1}(t)}{t_{i+k+1} - t_{i+1}} \right) \tag{5}$$

Remark 2. The derivative of a B-spline is also a B-spline of one less degree and it is given by

$$\boldsymbol{\phi}^{(j)}(t) = \sum_{i=0}^{n} \mathbf{p}_i N^{(j)}_{i,k}(t) \tag{6}$$

V. Approach

PINSAT interleaves *parallelized* discrete graph search in low-D with trajectory optimization in full-D to combine the benefits of the former's ability to search non-convex spaces and solve combinatorial parts of the problem and the latter's ability to obtain a locally optimal solution not constrained to discretization. In the following subsection, we will justify the choice of the low-D search algorithm. Subsequently, we will explain how an edge in the graph in $\mathcal{X}_L$ is lifted to $\mathcal{X}$ using B-spline optimization and the necessary modifications to the low-D search to make it compatible with PINSAT.

A. Low Dimensional Graph Search

The low-D search runs w-eA* [9] with variations that allow it to interleave with full-D trajectory optimization. Using w-eA* instead of wA* (as in INSAT) for the low-D search provides a systematic framework to parallelize expensive trajectory optimization with its parallelized variant w-ePA*SE. This is because, unlike in wA*, where the basic operation of the search loop is the expansion of a state, in w-eA* , the basic operation is *expansion of an edge*.

1) w-eA :* In w-eA* , the open list (*OPEN*) is a priority queue of edges (not states like in wA*) that the search has generated but has not expanded with the edge of least priority in front of the queue. The priority of an edge is $f((\mathbf{x}, \mathbf{a})) = g(\mathbf{x}) + h(\mathbf{x})$. Expansion of an edge $(\mathbf{x}, \mathbf{a})$ involves evaluating the edge to generate the successor $\mathbf{x}'$ and adding/updating (but not evaluating) the edges originating from $\mathbf{x}'$ into *OPEN* with the same priority of $g(\mathbf{x}') + h(\mathbf{x}')$. Henceforth, whenever $g(\mathbf{x}')$ changes, the positions of all of the outgoing edges from $\mathbf{x}'$ need to be updated in *OPEN*. To avoid this, ePA*SE replaces all outgoing edges of $\mathbf{x}'$ with a single *dummy* edge $(\mathbf{x}', \mathbf{a^d})$, where $\mathbf{a^d}$ stands for a dummy action until the dummy edge is expanded. Every time $g(\mathbf{x}')$ changes, only the dummy edge's position has to be updated. When the dummy edge $(\mathbf{x}', \mathbf{a^d})$ is expanded, it is replaced by the outgoing real edges from $\mathbf{x}'$ in *OPEN*. The real edges are expanded when they are popped from *OPEN* by an edge expansion thread. This decoupling of the expansion of the outgoing edges from the expansion of their common parent is what enables its asynchronous parallelization.

B. B-Spline Optimization

This subsection explains the abstract optimization over the length and duration of B-splines that satisfy the obstacle, duration, derivative, and boundary constraints. In the later section, we describe how it is utilized within PINSAT's parallel low-D graph search.

1) Decoupling Optimization over Trajectory and its Duration: It is easy to observe from Eq. 1 that there is a coupling between the trajectory $\boldsymbol{\phi}(t)$ and its duration T. For simultaneous optimization over the $\boldsymbol{\phi}$ and T, we decouple them by introducing a trajectory coordinate $u \in [0, 1]$ and a mapping to convert this coordinate to the actual time, $t = \alpha(u)$ [38]. Let the mapping α be monotonically increasing with $\alpha(0) = 0$ and $\alpha(1) = T$ and $\boldsymbol{\phi}(t) = \boldsymbol{\phi}(\alpha(u)) = (\boldsymbol{\phi} \circ \alpha)(u) = \boldsymbol{\psi}(u)$. The derivatives of the trajectory are

$$\boldsymbol{\phi}^{(1)}(t) = \frac{d\boldsymbol{\phi}(\alpha(u))}{d\alpha} \frac{d\alpha(u)}{du} \frac{du}{dt} = \frac{d\boldsymbol{\psi}(u)}{du} \frac{du}{dt} \tag{7}$$

Dropping the arguments for simplicity

$$\boldsymbol{\phi}^{(2)}(t) = \frac{d\boldsymbol{\psi}^2}{du^2}\left(\frac{du}{dt}\right)^2 + \frac{d\boldsymbol{\psi}}{ds}\frac{d^2u}{dt^2} \tag{8}$$

Similarly

$$\boldsymbol{\phi}^{(3)}(t) = \frac{d^3\boldsymbol{\psi}}{du^3}\left(\frac{du}{dt}\right)^3 + 3\frac{d^2\boldsymbol{\psi}}{du^2}\frac{d^2u}{dt^2}\frac{du}{dt} + \frac{d\boldsymbol{\psi}}{du}\frac{d^3u}{dt^3} \tag{9}$$

Algorithm 1 PINSAT: Planning Loop

$\mathcal{A} \leftarrow$ action space , $N_t \leftarrow$ number of threads, $G \leftarrow \emptyset$
$\mathbf{x}^S \leftarrow$ start state , $\mathcal{G} \leftarrow$ goal region, $terminate \leftarrow$ False
procedure PLAN()
  $\forall \mathbf{x} \in G$, $\mathbf{x}.g \leftarrow \infty$, $n\_successors\_generated(s) = 0$
  $\mathbf{x}^S.g \leftarrow 0$
  insert $(\mathbf{x}^S, \mathbf{a^d})$ in *OPEN* # Dummy edge from $\mathbf{x}^S$
  LOCK
  while not $terminate$ **do**
    if *OPEN* $= \emptyset$ **and** *BE* $= \emptyset$ **then**
      $terminate =$ True
      UNLOCK
      return $\emptyset$
    $(\mathbf{x}, \mathbf{a}) \leftarrow$ *OPEN*$.min()$
    if such an edge does not exist **then**
      UNLOCK
      wait until *OPEN* or *BE* change
      LOCK
      continue
    if $\mathbf{x} \in \mathcal{G}$ **then**
      $terminate =$ True
      UNLOCK
      return $\mathbf{x}.traj$
    else
      UNLOCK
      while $(\mathbf{x}, \mathbf{a})$ has not been assigned a thread **do**
        for $i = 1 : N_t$ **do**
          if thread i is available **then**
            if thread i has not been spawned **then**
              Spawn EDGEEXPANDTHREAD(i)
            Assign $(\mathbf{x}, \mathbf{a})$ to thread i
      LOCK
  $terminate =$ True
  UNLOCK

2) Transcription for Optimizing B-splines: The derivative of the B-spline curve $\boldsymbol{\phi}^{(j)}(t)$ in terms of the derivative of the B-spline basis $N_{i,k}^{(j)}(t)$ is given in Eq. 6. Using this relation the derivative of the B-spline curve with the trajectory coordinate $\boldsymbol{\psi}^{(j)}(u)$ in terms of the derivative's control points $\mathbf{p}_i^{(j)}$ can be derived as [42]

$$\boldsymbol{\psi}^{(j)}(u) = \sum_{i=0}^{n-j} \mathbf{p}_i^{(j)} N_{i,k-j}(u) \tag{10}$$

where $\mathbf{p}_i^{(j)}$ is given as

$$\mathbf{p}_i^{(j)} = \begin{cases} \mathbf{p}_i & j = 0 \\ \frac{k-j+1}{u_{i+k+1}-u_{i+j}}(\mathbf{p}_{i+1}^{(j-1)} - \mathbf{p}_i^{(j-1)}) & j > 0 \end{cases} \tag{11}$$

In this work, we use $t = \alpha(u) = Tu$. So $\frac{du}{dt} = \frac{1}{T}$ and $\frac{d^2u}{dt^2} = \frac{d^3u}{dt^3} = 0$ and Eq. 7 - Eq. 9 becomes

$$\boldsymbol{\phi}^{(j)}(t) = \frac{1}{T^j}\frac{d^j\boldsymbol{\psi}}{du^j} \quad \text{where } j = \{1, 2, 3\} \tag{12}$$

Algorithm 2 PINSAT: Edge Expansion

procedure EDGEEXPANDTHREAD(i)
  while not $terminate$ **do**
    if thread i has been assigned an edge $(\mathbf{x}, \mathbf{a})$ **then**
      EXPAND $((\mathbf{x}, \mathbf{a}))$
procedure EXPAND$((\mathbf{x}, \mathbf{a}))$
  LOCK
  if $\mathbf{a} = \mathbf{a^d}$ **then**
    insert $\mathbf{x}$ in *BE*
    for $\mathbf{a} \in \mathcal{A}$ **do**
      $f((\mathbf{x}, \mathbf{a})) = g(\mathbf{x}) + h(\mathbf{x})$
      insert $(\mathbf{x}, \mathbf{a})$ in *OPEN* with $f((\mathbf{x}, \mathbf{a}))$
  else
    UNLOCK
    $\mathbf{x}', c((\mathbf{x}, \mathbf{a})) \leftarrow$ GENERATESUCCESSOR $((\mathbf{x}, \mathbf{a}))$
    LOCK
    if $\mathbf{x}' \notin$ *CLOSED* $\cup$ *BE* **then**
      $\boldsymbol{\phi}_{\mathbf{x}^S\mathbf{x}'} =$ GENERATETRAJECTORY$(\mathbf{x}, \mathbf{x}')$
      if $g(\mathbf{x}') > c(\boldsymbol{\phi}(\mathbf{x}^S\mathbf{x}'))$ **then**
        $g(\mathbf{x}') = c(\boldsymbol{\phi}(\mathbf{x}^S\mathbf{x}'))$
        $\mathbf{x}'.parent = \mathbf{x}$
        $\mathbf{x}'.traj = \boldsymbol{\phi}_{\mathbf{x}^S\mathbf{x}'}$
        $f((\mathbf{x}', \mathbf{a^d})) = g(\mathbf{x}') + h(\mathbf{x}')$
        insert/update $(\mathbf{x}', \mathbf{a^d})$ in *OPEN* with $f((\mathbf{x}', \mathbf{a^d}))$
    $n\_successors\_generated(\mathbf{x}) += 1$
    if $n\_successors\_generated(\mathbf{x}) = |\mathcal{A}|$ **then**
      remove $\mathbf{x}$ from *BE*
      insert $\mathbf{x}$ in *CLOSED*
  UNLOCK

Now we can apply the above decoupling to the optimization in Eq. 1 with the new trajectory coordinate u and the map $\boldsymbol{\psi}$. Given two boundary states $\mathbf{x}', \mathbf{x}''$, Eq. 1 can be transcribed into a parameter optimization problem over the control points of the B-spline trajectory $\mathbf{P} = \{\mathbf{p}_0, \ldots, \mathbf{p}_m\}$ and the duration of the trajectory T using Eq. 12 as

$$\min_{\mathbf{p}_0,\ldots,\mathbf{p}_m,T} \quad w_1 T + w_2 \sum_{i=0}^{m-1} \left\|\mathbf{p}_{i+1} - \mathbf{p}_i\right\|_2 \tag{13a}$$

$$\text{s.t.} \quad 0 < t_{\min} \le T \le t_{\max} \tag{13b}$$

$$\mathbf{p}_0 = \mathbf{x}', \mathbf{p}_m = \mathbf{x}'' \tag{13c}$$

$$|\mathbf{p}_i^{(j)}| \preceq T^j \boldsymbol{\phi}_{\lim}^{(j)}; i = \{0, \ldots, m\}, j = \{1, 2, 3\} \tag{13d}$$

$$\boldsymbol{\psi}_{\mathbf{P},T}(u) \in \mathcal{X}^{\text{free}}; u \in [0, 1] \tag{13e}$$

where the constraint Eq. 13d comes from Eq. 12 and $\boldsymbol{\psi}_{\mathbf{P},T}$ is the trajectory reconstructed during every iteration using $\mathbf{P}$ and T as explained below. From Remark 1 [43], we know that the B-spline curve is entirely contained inside the convex hull of its control points. We leverage this property to formulate Eq. 13a and Eq. 13d. So it is enough to just enforce the trajectory derivative constraints at the control points (Eq. 13d) instead of checking if the curve satisfies them for all u (and therefore for all t).

Remark 3. The constraint on the control points in Eq. 13d is a sufficient but not necessary to enforce the limits on derivatives. In other words, it is possible to have control points outside the limits $\pm T^j \boldsymbol{\phi}_{\lim}^{(j)}$ and still have the curve satisfy these limits.

Note that Eq. 13 is a nonlinear program (NLP) for $j > 1$ as it has quadratic (or cubic, quintic, etc) constraints in the decision variable T (Eq. 13d). However, the following relaxation obtained by fixing the duration to $t_{\max}$ is convex

if $\mathcal{X}^{\text{free}}$ is convex

$$\min_{\mathbf{p}_0,\dots,\mathbf{p}_m} \quad w \sum_{i=0}^{m-1} \left\|\mathbf{p}_{i+1} - \mathbf{p}_i\right\|_2^2 \tag{14a}$$

$$\text{s.t. } \mathbf{p}_0 = \mathbf{x}', \mathbf{p}_m = \mathbf{x}'' \tag{14b}$$

$$|\mathbf{p}_i^{(j)}| \preceq t_{\max}^j \boldsymbol{\phi}_{\lim}^{(j)}; i = \{0,\dots,m\}, j = \{1,2,3\} \tag{14c}$$

$$\boldsymbol{\phi}_{\mathbf{P},t_{\max}}(t) \in \mathcal{X}^{\text{free}}; t \in [0, t_{\max}] \tag{14d}$$

3) B-spline Trajectory Reconstruction: Once the control point vector $\mathbf{P}$ and the duration of trajectory T are found by solving Eq. 13, the trajectory and its derivatives can be computed given the choice of B-spline basis $N_{i,k}(t)$ (Eq. 2) and the knot vector $\mathbf{T}$ (Eq. 3a) using Eq. 4 and Eq. 10. We also use this reconstruction to validate Eq. 13e within every iteration of the solver routine for obstacle avoidance, early termination and constraint satisfaction.

C. PINSAT: Parallelized Interleaving of Search And Trajectory Optimization

The pseudocode of PINSAT is given in Alg. 1 along with a graphical illustration in Fig. 2. The caption in Fig. 2 is intentionally detailed and not repeated in this section. In w-ePA*SE, to maintain bounded suboptimality, an edge can only be expanded if it is *independent* of all edges ahead of it in *OPEN* and the edges currently being expanded, i.e. in set *BE* (line 9) [9]. However, since INSAT and, therefore, PINSAT are not bounded suboptimal algorithms, the expensive independence check can be eliminated. Additionally, this increases the amount of parallelization that can be achieved. Alg. 1 begins by initializing g-values and inserting the start state with the dummy edge $(\mathbf{x}^S, \mathbf{a}^{\mathbf{d}})$ in *OPEN* (line 4-6). Once an edge is popped from *OPEN* (line 13), it is delegated for expansion to a spawned edge expansion thread. If all of the spawned edges are busy, a new thread is spawned as long as the total number of threads does not exceed the thread budget N_t (line 29). When N_t is higher than the number of edges available for expansion at any point in time, only a subset of available threads are spawned. This prevents performance degradation due to the operating system overhead of managing unused threads. The search terminates when the edge picked for expansion originates from a state contained in the goal region (line 19)

The pseudocode for asynchronous edge expansion delegated from the main search in Alg. 1 is given in Alg. 2. If a dummy edge is selected for expansion, then it is first replaced by unevaluated real edges in *OPEN* using the low-D action set $\mathcal{A}$ (line 7-11) and the priority value of its source state. Otherwise, the successor edge is generated and if this edge is neither in *CLOSED* nor in *BE* (line 16), the algorithm enters the trajectory optimization routine (line 12-27).

The trajectory optimization routine (Alg. 3) receives the set of ancestors of the edge being expanded (line 2). We solve the optimization problem described in the previous sections (Eq. 14) to find a corresponding full-dimensional trajectory from the start to the successor. Note that the optimization routine has access to the collision checker (implemented through callback mechanisms) and runs until convergence or collision occurrence, whichever comes first. The optimization iterates are reconstructed and checked for collision. In the case of collision, the best iterate that satisfies all the constraints is returned as a solution. This optimization is carried out in two steps. Only the trajectory generation between the ancestor node and the successor node, which is typically of very short-horizon, must be optimized from scratch (line 3, 9). Finding the entire trajectory to the successor from the start state can be warm-started (line 5, 10) using the incoming trajectory to the ancestor. This is possible as a result of the dynamic programming in the low-D search which guarantees that an edge out of node which does not contain a valid incoming trajectory will never be expanded. In heuristic search, edge evaluation refers to computing the cost and representation of an edge that is part of the underlying graph. The optimization step of rewiring to a better parent node, common in many search algorithms, is restricted to the edges within the graph. In contrast, a key distinction in INSAT and therefore PINSAT is that it optimizes for trajectories between nodes not connected by an edge in the low-D graph when searching for a full-D trajectory to a node. INSAT generates this *runtime* edge by searching over the set of ancestor nodes that lead to the node picked for expansion. Consequently, to make w-eA* and ePA*SE compatible with PINSAT, we search the set of ancestor edges leading to the edge chosen for expansion. If the optimized trajectory is invalid, the algorithm moves to the next ancestor (line 2).

Algorithm 3 PINSAT: Trajectory Optimization

```
1: procedure GENERATETRAJECTORY(x, x')
2:   for x'' ∈ ANCESTORS(x) ∪ x do                    # From x^S to x
3:     φ_{x''x'} = OPTIMIZE(x'', x')                  # Eq. 13
4:     if φ_{x''x'} is collision free then
5:       φ_{x^S x'} = WARMOPTIMIZE(φ_{x^S x''}, φ_{x''x'})   # Eq. 13
6:       return φ_{x^S x'}
7:     else if x'' = x then
8:       if φ^{t_max}_{x''x'} exists then              # Solve Eq. 14
9:         φ_{x''x'} = OPTIMIZE(x'', x')               # Eq. 13 with k_min
10:        φ_{x^S x'} = WARMOPTIMIZE(φ_{x^S x''}, φ_{x''x'})   # 13
11:        return φ_{x^S x'}
12:    return NULL
```

D. Theoretical Analysis

We begin with some preliminaries to show that PINSAT is a provably complete algorithm, *i.e.* it finds a solution if one exists.

Definition 1. Tunnel around Low-D Edges and Paths: For an edge $e \in \mathcal{E}$ in G, consider a low-D subspace corresponding to that edge called low-D tunnel of the edge $\tau_L(e) \subset \mathcal{X}_L$. The full-D tunnel for that edge is given by $\tau(e) = \tau_L(e) \times \mathcal{X}_A$. This definition can be extended to paths on the graph. Let a path between two nodes $\mathbf{x}', \mathbf{x}'' \in \mathcal{V}$ be given as $\boldsymbol{\theta}_{\mathbf{x}'\mathbf{x}''} = \{(\mathbf{x}^0, \mathbf{x}^1), (\mathbf{x}^1, \mathbf{x}^2), \dots, (\mathbf{x}^{W-1}, \mathbf{x}^W) \mid \mathbf{x}^0 = \mathbf{x}' \wedge \mathbf{x}^W = \mathbf{x}'' \wedge (\mathbf{x}^{w-1}, \mathbf{x}^w) \in \mathcal{E}, 1 \le w \le W\}$. Then a tunnel around $\boldsymbol{\theta}_{\mathbf{x}'\mathbf{x}''}$ is given as $\tau(\boldsymbol{\theta}_{\mathbf{x}'\mathbf{x}''}) = \cup_{e \in \boldsymbol{\theta}_{\mathbf{x}'\mathbf{x}''}} \tau(e)$.

Assumption 1. We assume that there exists at least a path on the low-D graph G from $\mathbf{x}^S$ to $\mathbf{x}^G$ such that its full-D tunnel contains $\boldsymbol{\phi}_{\mathbf{x}^S\mathbf{x}^G}$, *i.e.* $\exists \boldsymbol{\theta}_{\mathbf{x}^S\mathbf{x}^G} \in G \mid \exists \boldsymbol{\phi}_{\mathbf{x}^S\mathbf{x}^G} \in \tau(\boldsymbol{\theta}_{\mathbf{x}^S\mathbf{x}^G})$.

Assumption 2. $\forall e \in \mathcal{E}$, $\tau(e)$ is convex.

1) Minimum Order of B-spline: The minimum order $k_{\min}$ of the B-spline $\boldsymbol{\phi}_{k_{\min}}(t)$ with control points $\mathbf{P} = \{\mathbf{p}_0, \dots, \mathbf{p}_m\}$ that is (i) guaranteed to lie entirely within the full-D tunnel of any edge $e = (\mathbf{x}', \mathbf{x}'') \in \mathcal{E}$ and (ii) satisfy constraints on its derivatives (similar to Eq. 13d) can

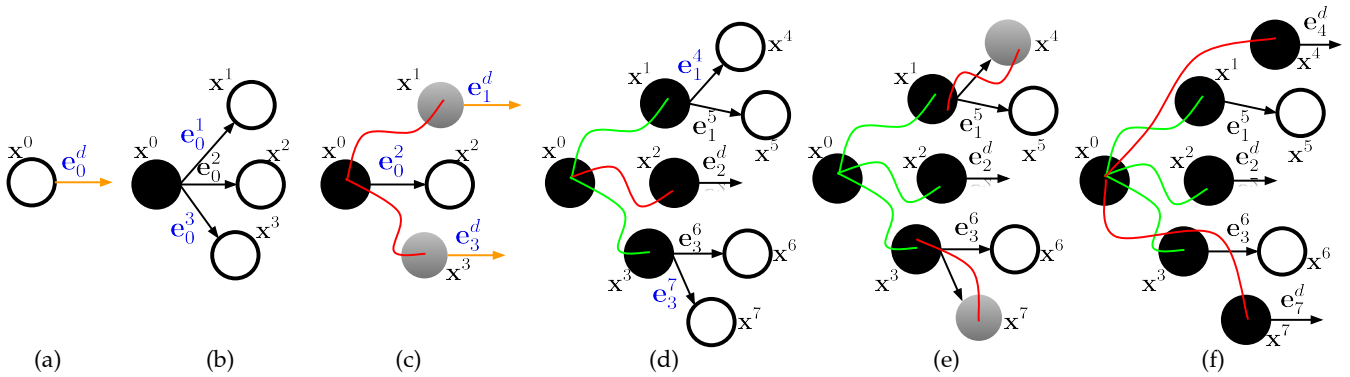

Fig. 2: Graphical illustration of PINSAT. (Fig. a, b) The dummy edge e_0^d from $\mathbf{x}^0$ is expanded to get real low-D edges $[e_0^1, e_0^2, e_0^3]$ but not yet evaluated with the optimizer in full-D to generate the state successor (denoted with hollow node). (c) The ePA*SE architecture in PINSAT then evaluates multiple edges in parallel (red curves). Once evaluated the outgoing nodes $\mathbf{x}^1$&$\mathbf{x}^3$ are represented with dummy edges e_1^d & e_3^d and inserted in OPEN. This node is shown with a gray gradient to represent the underspecified full-D state for the optimization to figure out. In Fig. (d), we highlight the asynchronous execution of PINSAT. Here the dummy edges e_1^d & e_3^d are expanded into real edges and the e_0^2 is lifted to full-D by running the optimizer. Similarly, in Fig. (e), more edges are concurrently lifted to full-D. Fig. (f) denotes how the short incremental trajectory generated in Fig. (e) is reused for warm-starting the optimization and generating the trajectories from $\mathbf{x}^S$. This is a powerful step that dramatically increases the speed of convergence. Note that once the trajectory from $\mathbf{x}^S$ is generated, it replaces the low-D edge and helps drive the graph search into more informative directions.

be found by solving the simple optimization below

$$
\begin{aligned}
k_{\min} = \quad & \max_{e \in \mathcal{E}} \min_{k_e} \quad k_e && \text{(15a)}\\
\text{s.t.} \quad & \mathbf{p}_0 = \mathbf{x}', \mathbf{p}_m = \mathbf{x}'' && \text{(15b)}\\
& \boldsymbol{\phi}_{k_{\min}}(t) \in \tau_L(e); 0 \le t \le t_{\min} && \text{(15c)}\\
& |\mathbf{p}_0^{(j)}|, |\mathbf{p}_{k_e-j}^{(j)}| = t_{\min}^j \boldsymbol{\phi}_{\lim}^{(j)} && \text{(15d)}\\
& |\mathbf{p}_1^{(j)}|, \ldots, |\mathbf{p}_{k_e-j-1}^{(j)}| \le t_{\min}^j \boldsymbol{\phi}_{\lim}^{(j)} && \text{(15e)}
\end{aligned}
$$

The tunnel constraints in Eq $15c$ are transcribed into constraints on the control points similar to how corridor constraints are represented in [44]. In this work, the outer maximization is carried over actions set $\mathcal{A}$, which is typically a significantly smaller set than $\mathcal{E}$. This can be done if the edges in the graph can be grouped as a finite number of action primitives sharing the same tunnel representation. Note that the inner optimization in the above formulation is a linear program and is guaranteed to have at least one solution as long as the feasible region of the decision variable is nonempty and bounded.

Remark 4. The feasible region depends on the limits of the system $\boldsymbol{\phi}_{\lim}^{(j)}$ and the choice of the tunnel τ which is domain-specific. However, in most cases, a straightforward design of the tunnel [44] can ensure a nonempty feasible region and a reasonably low $k_{\min}$.

Consider an edge $e = (\mathbf{x}', \mathbf{x}'') \in \mathcal{E}$.

Lemma 1. If $\tau(e)$ is convex, then a $\boldsymbol{\phi}^{\sim}_{\mathbf{x}'\mathbf{x}''}(t) \in \tau(e), \forall t \in [0, t_{\max}]$ that solves Eq. 14 to global optimality can be found.

Proof. Setting $\mathcal{X}^{\text{free}} = \tau(e)$ make Eq. 14d a convex domain constraint on the decision variables and Eq. 14 a convex program. Assuming $t_{\max}$ is large enough, we know that a convex optimization can be solved to global optimality. □

Let $\hat{\boldsymbol{\phi}}_{\mathbf{x}'\mathbf{x}''} \in \mathcal{C}^{\gamma}(\mathbb{R})$. $\hat{\boldsymbol{\phi}}_{\mathbf{x}'\mathbf{x}''}$ need not satisfy $\boldsymbol{\phi}_{\lim}^{(j)}, j \le \gamma$.

Lemma 2. If $\hat{\boldsymbol{\phi}}_{\mathbf{x}'\mathbf{x}''} \in \tau_L(e)$, then there exists a $k_{\min}$-th order $\boldsymbol{\phi}^*_{\mathbf{x}'\mathbf{x}''} \in \tau(e)$ that satisfies Eq. 13.

Proof. It directly follows from the constraint satisfaction for finding $k_{\min}$ in Eq. 15. This $k_{\min}$ can be used to choose the B-spline basis for optimizing Eq. 13 simultaneously over path length and time to find $\boldsymbol{\phi}^*_{\mathbf{x}'\mathbf{x}''} \in \tau(e)$. □

In other words, Lemma. 2 states that if there is a trajectory that is (i) $\mathcal{C}^j$-continuous, (ii) satisfies the low-D boundary conditions of an edge $e \in \mathcal{E}$ and, (iii) is contained entirely within the low-D tunnel of the edge $\tau_L(e)$, then we can use $k_{\min}$-th order basis functions to construct a trajectory that satisfies full-D j-th derivative boundary conditions and lies entirely within the full-D tunnel of the edge $\tau(e)$. We note that Lemma. 2 is a critical building block towards developing PINSAT's guarantee on completeness.

Theorem 1. Completeness of PINSAT: If Assumption 1 holds, then PINSAT with $k_{\min}$-th order B-spline optimization is guaranteed to find a $\boldsymbol{\phi}_{\mathbf{x}^S\mathbf{x}^G}$ that satisfies Eq. 1.

VI. Evaluation

We evaluated PINSAT by evaluating it on two kinodynamic manipulation planning scenarios: (i) a single ball blocking task among obstacles using a 6 DoF ABB arm, and (ii) a multi-ball blocking task where the balls are separated by short time intervals using a 7 DoF KUKA LBR iiwa arm with obstacles. All experiments were run on an AMD Ryzen Threadripper Pro with 128 threads.

A. Single Ball Blocking with ABB IRB 1600

The ABB IRB 1600 industrial robot used for this experiment is shown in Fig 1. The red horizontal and vertical bars are obstacles and divide the region around the robot into eight quadrants, four below the horizontal bars and four above. We randomly sample 500 hard start and goal configurations. We ensure that the end effector corresponding to one of those configurations is above the horizontal bars and the other is underneath. We also ensure that the end effector for the start and goal configurations is not sampled in the region enclosed by the same two adjacent vertical bars. Our samples are deliberately far apart in the manipulator's C-space and require long spatial and temporal plans to connect them. The motivation to sample start-goal pairs in this manner is to

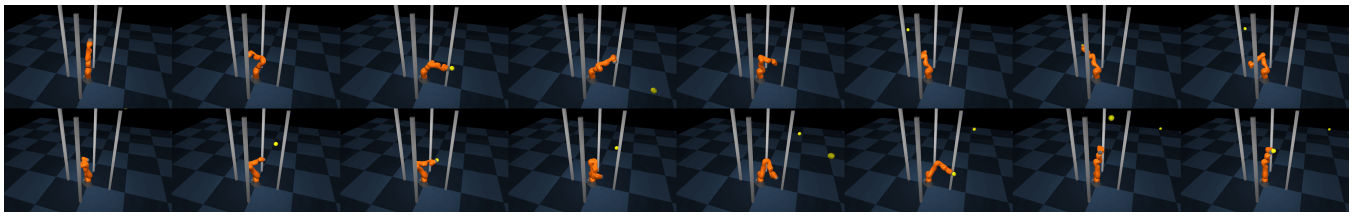

Fig. 3: Multi-ball blocking scenario demonstrated using KUKA LBR iiwa arm. In this scenario, five balls separated by short time intervals are thrown at the arm. The planner is given the exact time and location to block the ball. PINSAT finds a single smooth kinodynamic trajectory that reaches every ball with a predisposed joint velocity suitable to reach the subsequent ball right on time.

	pBiRRT +TrajOpt				w-ePA*SE +TrajOpt				INSAT	PINSAT			
Threads	5	10	50	120	5	10	50	120	1	5	10	50	120
Success rate (%)	1	1	3	2	0	0	6	2	58	72	81	90	90
Time (s)	0.03±0.02	0.021±0.02	0.02±0.01	0.02±0.01	0.38±0.35	0.30±0.31	0.28±0.32	0.27±0.30	2.02±2.01	0.70±0.91	0.43±0.6	0.43±0.31	0.99±0.72
Cost	5.18±2.89	4.88±2.75	4.55±2.48	4.25±2.24	9.19±4.23	8.71±4.00	8.34±3.89	8.23±3.76	1.55±2.68	0.51±1.11	1.10±2.06	1.47±2.54	1.57±2.62

TABLE I: Mean and standard deviation of planning time and cost for PINSAT and the baselines for different thread budgets.

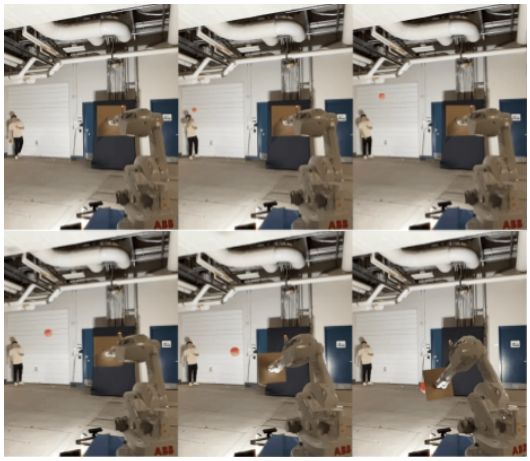

Fig. 4: PINSAT used to accelerate preprocessing of a motion library for rapid lookup. Here, an ABB arm with a shield attached to its end-effector executes a motion generated by PINSAT to block a ball launched toward it.

subject the planner to stress tests under severe constraints on the duration of the trajectory.

For our experiments, we use a backward breadth-first search (BFS) from the goal state in the task space of the manipulator as our heuristic. The action space $\mathcal{A}$ is made up of small units of joint movements. For every joint, we had a $4\,\mathrm{deg}$ and $7\,\mathrm{deg}$ primitive in each direction subject to the joint limits. In addition to this, during every state expansion, we also check if the goal state is the line of sight to consider the goal as a potential successor. The velocity limits of the arm were inflated 10x the values specified in the datasheet for simulation. We used $50m/s^2$ and $200m/s^3$ as acceleration and jerk limits. The maximum duration of the trajectory was limited to 0.6s. The reason for inflating the velocity limits is to allow for the bare minimum time to navigate the obstacles and reach from start to goal.

Table I shows success rate and planning time statistics for INSAT and PINSAT for different thread budgets. The planning time statistics were only computed over the problems successfully solved by both algorithms for a given thread budget. We also implemented a parallelized version of RRTConnect (pBiRRT) as a baseline. The kinematic plan generated by pBiRRT was then post-processed by the same B-spline optimization used in PINSAT to compute the final kinodynamic plan. As the optimization has a constraint on the maximum execution duration of the trajectory, adding all the waypoints from the path at once will result in an over-constrained system of equations for which a solution may not exist. To circumvent this, we iteratively add the waypoint constraint starting from the minimum containing just the start and the goal state. Though pBiRRT had a 100% success rate in computing the spatial trajectory, the results after post-processing with the optimizer were abysmal. PINSAT achieves a significantly higher success rate than INSAT for all thread budgets greater than 1. Specifically, it achieves a 5x improvement in mean planning time, a 7x improvement in median planning time, and a 1.8x improvement in success rate for $N_t = 50$. Even with a single thread, PINSAT achieves planning times lower than INSAT as a result of decoupling edge evaluations from state expansions in ePA*SE.

B. Multi-Ball Blocking with KUKA iiwa

In this experiment, a sequence of balls separated by varying short time intervals is launched into the robot. The task is to generate a single smooth kinodynamic trajectory that blocks any of the balls from reaching the robot base. Only PINSAT successfully found a solution for this scenario as the baselines do not plan with time. As a result, the baselines were not able to reach the ball locations in the right time to intercept. We tried different post-processing and profiling approaches for w-ePA*SE and pBiRRT without any success. PINSAT was able to generate a single motion that intercepted all balls at the right time. One interesting observation is that the solution trajectory of PINSAT had an appropriate direction of the joint velocity in intercepting a ball such that it could reach the subsequent ball at the right time.

VII. Conclusion

We presented PINSAT, an extension of INSAT that asynchronously parallelizes the numerous *expensive* calls to the trajectory optimizer. PINSAT achieves this by incorporating the concepts of edge expansion and asynchronous edge evaluation borrowed from the recently introduced parallel search algorithm ePA*SE [9]. We evaluated our algorithm using a kinodynamic manipulation planning domain and demonstrated significantly higher success rates than INSAT [6], [7], along with a drastic reduction in planning time.

PINSAT guarantees only completeness at present, but in the future, we aim to explore how to establish independence checks, as done in ePA*SE, to guarantee optimality.

VIII. Acknowledgements

This work was supported by NSF Award CMMI-1734360-NSF entitled "NRI: INT: COLLAB: In-Situ Collaborative Robotics in Confined Spaces" with Nabil Simaan at Vanderbilt University and grants W911NF-18-2-0218 and W911NF-21-1-0050 of the ARL-sponsored A2I2 program.

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, 1968.

[2] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial intelligence*, vol. 1, no. 3-4, pp. 193–204, 1970.

[3] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, "Multi-heuristic a*," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 224–243, 2016.

[4] T. Kusnur, S. Mukherjee, D. M. Saxena, T. Fukami, T. Koyama, O. Salzman, and M. Likhachev, "A planning framework for persistent, multi-uav coverage with global deconfliction," in *Field and Service Robotics*, pp. 459–474, Springer, 2021.

[5] S. Mukherjee, C. Paxton, A. Mousavian, A. Fishman, M. Likhachev, and D. Fox, "Reactive long horizon task execution via visual skill and precondition models," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5717–5724, IEEE, 2021.

[6] R. Natarajan, H. Choset, and M. Likhachev, "Interleaving graph search and trajectory optimization for aggressive quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5357–5364, 2021.

[7] R. Natarajan, G. L. Johnston, N. Simaan, M. Likhachev, and H. Choset, "Torque-limited manipulation planning through contact by interleaving graph search and trajectory optimization," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8148–8154, IEEE, 2023.

[8] R. Natarajan, G. L. Johnston, N. Simaan, M. Likhachev, and H. Choset, "Long-horizon torque-limited planning through contact using discrete search and continuous optimization," in *IROS 2023 Workshop on Leveraging Models for Contact-Rich Manipulation*, 2023.

[9] S. Mukherjee, S. Aine, and M. Likhachev, "ePA*SE: Edge-based parallel A* for slow evaluations," in *International Symposium on Combinatorial Search*, vol. 15, pp. 136–144, AAAI Press, 2022.

[10] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

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

[12] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[13] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *Int. J. Robot. Research*, vol. 20, no. 5, pp. 378–400, 2001.

[14] T. Kunz and M. Stilman, "Kinodynamic rrts with fixed time step and best-input extension are not probabilistically complete," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pp. 233–244, Springer, 2015.

[15] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1431–1443, 2016.

[16] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[17] 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.

[18] M. Toussaint, "A tutorial on newton methods for constrained trajectory optimization and relations to slam, gaussian process smoothing, optimal control, and probabilistic inference," *Geometric and numerical foundations of movements*, pp. 361–392, 2017.

[19] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1168–1175, IEEE, 2014.

[20] 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)*, pp. 3067–3074, IEEE, 2015.

[21] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, "Sampling-based optimal kinodynamic planning with motion primitives," *Autonomous Robots*, vol. 43, no. 7, pp. 1715–1732, 2019.

[22] Z. Littlefield and K. E. Bekris, "Efficient and asymptotically optimal kinodynamic motion planning via dominance-informed regions," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, IEEE, 2018.

[23] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer, "Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4207–4214, IEEE, 2016.

[24] J. Kamat, J. Ortiz-Haro, M. Toussaint, F. T. Pokorny, and A. Orthey, "Bitkomo: Combining sampling and optimization for fast convergence in optimal motion planning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4492–4497, IEEE, 2022.

[25] K. V. Alwala and M. Mukadam, "Joint sampling and trajectory optimization over graphs for online motion planning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4700–4707, IEEE, 2021.

[26] J. Ortiz-Haro, W. Hoenig, V. N. Hartmann, and M. Toussaint, "idb-a*: Iterative search and optimization for optimal kinodynamic motion planning," *arXiv preprint arXiv:2311.03553*, 2023.

[27] S. A. Jacobs, K. Manavi, J. Burgos, J. Denny, S. Thomas, and N. M. Amato, "A scalable method for parallelizing sampling-based motion planning algorithms," in *2012 IEEE International Conference on Robotics and Automation*, pp. 2529–2536, 2012.

[28] D. Devaurs, T. Siméon, and J. Cortés, "Parallelizing rrt on distributed-memory architectures," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2261–2266, 2011.

[29] J. Ichnowski and R. Alterovitz, "Parallel sampling-based motion planning with superlinear speedup.," in *IROS*, pp. 1206–1212, 2012.

[30] S. A. Jacobs, N. Stradford, C. Rodriguez, S. Thomas, and N. M. Amato, "A scalable distributed rrt for motion planning," in *2013 IEEE International Conference on Robotics and Automation*, pp. 5088–5095, 2013.

[31] J. Liang, M. Sharma, A. LaGrassa, S. Vats, S. Saxena, and O. Kroemer, "Search-based task planning with learned skill effect models for lifelong robotic manipulation," *arXiv preprint arXiv:2109.08771*, 2021.

[32] M. Phillips, M. Likhachev, and S. Koenig, "Pa* se: Parallel a* for slow expansions," in *Twenty-Fourth International Conference on Automated Planning and Scheduling*, 2014.

[33] S. Mukherjee, S. Aine, and M. Likhachev, "Mplp: Massively parallelized lazy planning," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6067–6074, 2022.

[34] Y. Zhou and J. Zeng, "Massively parallel a* search on a gpu," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

[35] X. He, Y. Yao, Z. Chen, J. Sun, and H. Chen, "Efficient parallel a* search on multi-gpu system," *Future Generation Computer Systems*, vol. 123, pp. 35–47, 2021.

[36] L. Li, Y. Miao, A. H. Qureshi, and M. C. Yip, "Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4496–4503, 2021.

[37] S. Chow, D. Chang, and G. A. Hollinger, "Parallelized control-aware motion planning with learned controller proxies," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2237–2244, 2023.

[38] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2427–2433, IEEE, 2009.

[39] H. Liu, X. Lai, and W. Wu, "Time-optimal and jerk-continuous trajectory planning for robot manipulators with kinematic constraints," *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 2, pp. 309–317, 2013.

[40] P. Kicki, P. Liu, D. Tateo, H. Bou-Ammar, K. Walas, P. Skrzypczyński, and J. Peters, "Fast kinodynamic planning on the constraint manifold with deep neural networks," *arXiv preprint arXiv:2301.04330*, 2023.

[41] N. M. Patrikalakis and G. A. Kriezis, "Representation of piecewise continuous algebraic surfaces in terms of b-splines," *The visual computer*, vol. 5, pp. 360–370, 1989.

[42] L. Piegl and W. Tiller, *The NURBS book*. Springer Science & Business Media, 1996.

[43] C. De Boor, *A practical guide to splines*, vol. 27. springer-verlag New York, 1978.

[44] G. Rousseau, C. S. Maniu, S. Tebbani, M. Babel, and N. Martin, "Minimum-time b-spline trajectories with corridor constraints. application to cinematographic quadrotor flight plans," *Control Engineering Practice*, vol. 89, pp. 190–203, 2019.