2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Madrid, Spain, October 1-5, 2018

Leveraging Precomputation with Problem Encoding for Warm-Starting Trajectory Optimization in Complex Environments

Wolfgang Merkt, Vladimir Ivan, and Sethu Vijayakumar

***Abstract*— Motion planning through optimization is largely based on locally improving the cost of a trajectory until an optimal solution is found. Choosing the initial trajectory has therefore a significant effect on the performance of the motion planner, especially when the cost landscape contains local minima. While multiple heuristics and approximations may be used to efficiently compute an initialization online, they are based on generic assumptions that do not always match the task at hand. In this paper, we exploit the fact that repeated tasks are similar according to some metric. We store solutions of the problem as a library of initial seed trajectories offline and employ a problem encoding to retrieve near-optimal warm-start initializations on-the-fly. We compare how different initialization strategies affect the global convergence and runtime of quasi-Newton and probabilistic inference solvers. Our analysis on the 38-DoF NASA Valkyrie robot shows that efficient and optimal planning in high-dimensional state spaces is possible despite the presence of globally non-smooth and discontinuous constraints, such as the ones imposed by collisions.**

I. Introduction

Motion synthesis refers to the process of finding a collision-free trajectory or control policy for moving a robotic system to accomplish a given task. Expressive motions are composed of *constraints*, e.g., reaching a desired end-effector position, keeping a liquid-filled container level, avoiding obstacles, or maintaining balance on a bipedal robot, as well as *objectives*, e.g., smoothness, minimal energy use, and natural appearance. These desired properties can be captured in objective and constraint functions and formulated as an optimization problem. The solution to this problem can be a trajectory, i.e., a set of joint or state space keyframes or functions (trajectory optimization), or time-varying local feedback control laws (optimal control).

Computing optimal, collision-free motion and control policies for high-dimensional systems in complex environments is a time-consuming process, with the solutions often local to the initial conditions. Real-world environments feature clutter and come with complex, varied task goals, where tasks may change based on high-level user or perception input. Additionally, uncertainty in the perception or control of a system can lead to a divergence from the initial conditions present during planning. These changes can render previously computed solutions invalid requiring local adaptation or recomputation from scratch. Especially for complex tasks involving narrow passages and concave regions, cost functions often feature multiple local minima which may be time-consuming to find and hard to escape.

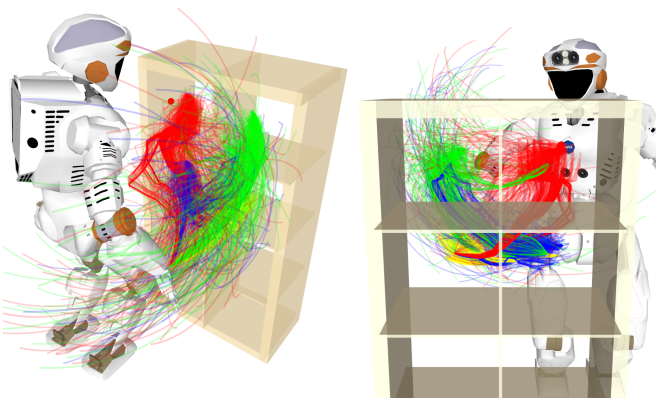

Fig. 1: Pick-and-place scenario in a complex environment with collision avoidance. A humanoid robot has to retrieve an item from a shelf, the target of which may change based on perception information or user input. In order to synthesize optimal motion fast online, we investigate different warm-start strategies for trajectory optimization methods.

Recently, trajectory optimization has been shown to scale to high-dimensional systems thanks to new formulations, more efficient solvers, and higher-performance computing equipment [1]–[3]. These fast optimization-based approaches generally require smooth, differentiable cost and objective functions, can take a long time to converge, and may get stuck in undesirable local minima—rendering good initialization a decisive factor for achieving high success rates and short planning times. Taking into account collision avoidance is non-trivial and may lead to highly nonlinear objective functions with many local minima. However, for most real-world applications such as in households or industrial human-robot collaboration, we see a rapid sequence of similar tasks in comparable environments.

In this work, we focus on providing good initialization seeds for on-the-fly optimal, collision-free motion synthesis on high-dimensional systems in complex and changing environments. We hereto introduce a problem encoding to build a trajectory library offline, into which we can index online to retrieve a warm-start seed. We detail our precomputation exploration strategy and explain how solution subindexing and goal region growing can be incorporated for efficient solution reuse and to reduce precomputation

This research is supported by the Engineering and Physical Sciences Research Council (EPSRC, grant reference EP/L016834/1) and EU H2020 project Memory of Motion (MEMMO, project ID: 780684). The work has been performed at the University of Edinburgh under the Centre for Doctoral Training in Robotics and Autonomous Systems program.

All authors are with the Institute for Perception, Action, and Behaviour, School of Informatics, The University of Edinburgh (Informatics Forum, 10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom). Email: wolfgang.merkt@ed.ac.uk.

and online memory requirements. The motion synthesis is formulated as a nonlinear optimization problem with first-order Markov dynamics. We provide analytic first-order derivatives approximating the collision avoidance objective as a local, virtual object penetration cost and compare quasi-Newton and variational inference optimization methods on the formulated problem. We then benchmark commonly used initialization strategies with our proposed method and compare their performance with regards to convergence, success rates, and use cases in a random, complex shelf picking scenario on a high-dimensional humanoid robot. A supplementary video is available at `https://youtu.be/Omg3FhVi_Tk`.

II. Related work

Common trajectory optimization formulations differ in their use of derivative and environment information as well initializations. Transcription methods convert the continuous optimal control problem into a nonlinear program (NLP) subject to equality and inequality constraints. If a derivative can be defined or approximated, unconstrained NLPs can be solved efficiently by Newton or quasi-Newton method algorithms which use second- and first-order information, respectively. Similarly, for formulations with hard constraints, Augmented Lagrangian, interior point, or sequential quadratic programming (SQP) methods can be applied. Recent gradient-based frameworks include CHOMP [4], TrajOpt [1], KOMO [2], and RieMo [5]. Stochastic, derivative-free algorithms which apply Monte Carlo methods on roll-outs include STOMP [6] and PI^2 [7].

AICO [8] formulates a probabilistic trajectory model and uses iterative message passing and approximate inference techniques to solve the nonlinear stochastic optimal control problem. It has been shown to be highly efficient as the variational message passing allows to update single states repeatedly without having to roll-out the entire trajectory.

As discontinuities and non-smooth cost gradients are difficult to handle, the way collision avoidance is included in the cost and constraint functions has received additional attention. TrajOpt [1] uses approximate convex decomposition and convex-convex collision checking penalizing penetrations with a hinge loss. CHOMP [4] and STOMP [6] use the Euclidean Distance Transform and overlapping sphere approximations for the robot body. As a consequence, these approaches require online or offline preprocessing of the environment and robot model. RieMo [5] applies Riemannian geometry to the workspace in order to plan motion in the presence of thin or long obstacles which translate into many local minima. As collision queries are expensive, [9] proposed an adaptive collision checking density with more checks closer to obstacles. The collision avoidance terms are still in general very likely to create local minima. To get around this locality issue, an initial trajectory should therefore lie in the same neighborhood as the global minimum.

In practice, motion optimization frameworks commonly use zero motion initialization (e.g., RieMo [5]), or straight-line interpolation between start and goal configurations assuming knowledge of a final configuration for instance from inverse kinematics (e.g., CHOMP [4], STOMP [6]). These initializations are often infeasible and a potentially lengthy stochastic search may be required to find a solution. Alternatively, two-phase optimization has been proposed where a first result is obtained using a simplified cost and the full optimization warm-started using this result [9]. Similar to model-predictive control, planning and execution can be interleaved by providing a fixed time budget for planning and warm-starting subsequent optimizations with the suboptimal solution from the previous timestep [10].

In order to speed up and ensure convergence of local optimization solvers, near-optimal initialization has been proposed. Here, the similarity to previously solved problems is used instead of planning from scratch [11]–[13]. For instance, trajectory libraries and function approximation have been considered to speed up online optimization [11], [12]. To predict warm-start seeds in cluttered environments, [13] focussed on learning expressive task and environment descriptors. Conversely, the work presented in [14] iteratively approximates the expensive-to-evaluate value and policy function using a neural network-based regression model employed as a distance metric and to evaluate transitions in a kinodynamic probabilistic roadmap (PRM).

On the other hand, sampling-based planners (SBP), such as Rapidly-Exploring Random Trees (RRT) or Probabilistic Road Maps (PRM), have been demonstrated to produce feasible motion plans in less than one second for high-dimensional systems in cluttered environments [15]. Constrained SBP algorithms can for instance take balancing into account and compute whole-body motion plans on humanoid robots in the order of seconds [16]. Recently, Hierarchical Dynamic Roadmaps (HDRM) which use a configuration-to-workspace-occupation encoding to offload collision checking to an offline precomputation phase have been shown to compute feasible trajectories for industrial manipulators in 1–2 ms [17]. These results encourage the use of SBPs for warm-starting optimization-based methods. However, by nature, geometric SBP methods produce a set of *kinematically feasible configurations*. These are often long, sub-optimal paths and may contain abrupt changes in velocities and accelerations. In order to be executable, i.e., dynamically feasible w.r.t. actuation constraints, post-processing such as time spacing, short-cutting, and smoothing is required. This does not, however, take into account *optimality* such as smoothness in higher-order terms (velocity, acceleration, jerk) or the dynamics of the plant (e.g., torque limits). While kinodynamic and combinations of optimization- and sampling-based planners (such as RRT^*) have been explored, additional temporal motion constraints such as balance and end-effector orientation during the motion are challenging using sampling-based algorithms alone as the sample adaptation step can limit exploration. We argue that SBPs may provide a feasible initialization in a local minimum for an optimization-based algorithm for one class of problems, but also propose an alternate method exploiting a database of previous solutions. To this end, we analyze different initialization strategies.

III. Problem Formulation

We consider a trajectory optimization problem with first-order Markov dynamics, similar to the unconstrained KOMO [2]. Here, we use a pseudo-dynamically weighted state transition cost $\ell_x(t) = (\mathbf{x}_t - \mathbf{x}_{t-1})^T W (\mathbf{x}_t - \mathbf{x}_{t-1})$ where W is a relative measure of the mass ratio of the kinematic chain that is moved by the respective joint: $W_{j,j} = \frac{\sum_{i=j}^{N} m_i}{M}$. After transcription, the synthesis of an optimal motion X^* in an environment Ω is an unconstrained nonlinear minimization problem over the evolution of system states $X = (\mathbf{x}_0, \cdots, \mathbf{x}_T)$, where the time horizon is uniformly discretized into T time steps:

$$X^* = \arg\min_X \sum_{t=1}^{T} [\ell_x(t) + \ell_y(t, \mathbf{y}^*)] \tag{1}$$

$$\text{with } \ell_y(t, \mathbf{y}^*) = \ell_{CoM} + \ell_{JointLimits} + \ell_{Position} + \tag{2}$$
$$+ \ell_{Orientation} + \ell_{CollisionAvoidance}$$

Each term in (2) is defined as a square cost $\rho_t \|\mathbf{y}_t^* - \phi(\mathbf{x}_t)\|^2$. Here, ϕ is a mapping from the configuration space to a task space with a task space goal $\mathbf{y}^*$ and ρ a mixing weight. The task space is commonly composed of forward kinematics, center of mass, or alternate spaces such as distance or interaction meshes [18], [19]. We use the following four types of task spaces in our experiments:

1) Center-of-Mass (CoM): To maintain static stability on flat terrain, the projection of the center-of-mass has to fall within the convex hull of its contact points (referred to as the support polygon). For quasi-static motions and to account for state estimation error, this support polygon is shrunk by a safety margin γ. Then, in order to favor stable configurations, a quadratic penalty for the deviation of the CoM-projection from the center of the support polygon is used.

2) Joint Limit: To avoid joint limits, we add a squared hinge loss to configurations that are within a set percentage of the bounds of the joint range.

3) Palm Position and Orientation: The position and orientation of the end-effector relative to the reaching goal frame as obtained through a forward kinematic map.

4) Collision Avoidance: In order to obtain a smooth, differentiable cost for collision avoidance, we compute a collision proxy $P_{A,B}$ for every pair of collision objects A, B which contains *a)* a signed distance $d_{A,B}$ for the closest distance or deepest penetration between the objects, *b)* the closest points between or the deepest penetration point on either body $\mathbf{p}_A, \mathbf{p}_B$, and *c)* the normals $\hat{\mathbf{n}}_A = \mathbf{p}_B - \mathbf{p}_A, \hat{\mathbf{n}}_B = \mathbf{p}_A - \mathbf{p}_B$ between these two virtual points:

$$P_{A,B} = (d_{A,B}, \mathbf{p}_A, \mathbf{p}_B, \hat{\mathbf{n}}_A, \hat{\mathbf{n}}_B) \tag{3}$$

The concept of using virtual collision proxies is illustrated in Figure 2 for two spheres as well as the shelf scenario considered (only external/interaction proxies are visualized).

Using the collision proxies for a pair of collision bodies, we can now formulate a smooth penalty for when the bodies

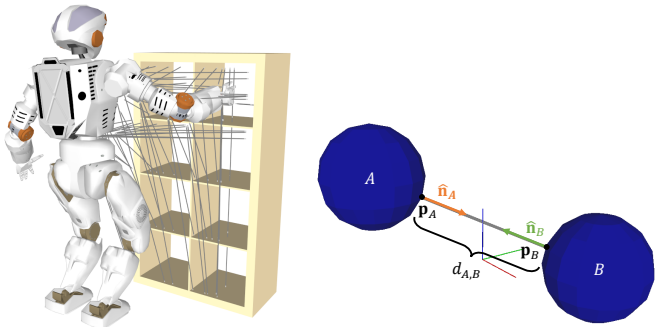

Fig. 2: Collision proxies: the grey line visualizes the normal between the nearest points on either collision body along which the virtual separation and penetration will move.

are closer than a threshold ϵ and 0 otherwise:[1]

$$\ell_{CollisionAvoidance}(\mathbf{x}, \Omega) = \sum_P \begin{cases} \left(1 - \frac{d}{\epsilon}\right)^2 & \text{if } d < \epsilon \\ 0 & \text{if } d \geq \epsilon \end{cases} \tag{4}$$

Suitable values for ϵ depend on the time discretization, the velocity limit, the accuracy of the tracking controller, and the target application. Assuming that moving along the normal between the contact/nearest points increases/decreases separation or penetration, we can formulate a derivative of the cost using the geometric Jacobian of the contact points on the collision bodies:

$$J_{CollisionAvoidance}(\mathbf{x}, \Omega) =$$
$$\sum_P \begin{cases} \frac{2}{\epsilon^2} \mathbf{n}_A \cdot J_{\mathbf{p}_A}(\mathbf{x}) - \frac{2}{\epsilon^2} \mathbf{n}_B \cdot J_{\mathbf{p}_B}(\mathbf{x}) & \text{if } d < \epsilon \\ 0 & \text{if } d \geq \epsilon \end{cases} \tag{5}$$

Similarly, we derive analytical first derivatives for all other cost terms using the geometric Jacobian and the chain rule.

Combinations of the first three terms in the objective function are nonlinear and non-convex but they are smooth, continuous, and with relatively few local minima. On the other hand, the collision avoidance cost is highly discontinuous because it depends heavily on the geometry of the environment. This often gives rise to multiple local minima. Warm-starting the optimization in the correct part of the space is therefore crucial.

IV. Warm-start Initialization

Since most of the complexity arises from the collision cost, we begin by finding a collision-free trajectory that does not have to satisfy any other optimality criteria. Hereto, we use a sampling-based planning algorithm to obtain a sequence of valid, i.e., collision-free and balanced configurations. In a post-processing step, smoothing and short-cutting are applied and timing adjusted so that the velocity limit constraints are met. Afterwards, the time-spaced sequence of configurations is sub-sampled using spline interpolation to create a uniformly spaced trajectory. In order to verify that the interpolated trajectory continues to be collision-free, further collision-checking is performed prior to being used as a seed trajectory. Thus, the feasible seed trajectory time horizon T is determined

[1]This proxy cost term and its first-order derivative are based on the source code implementation of AICO as used in [19].

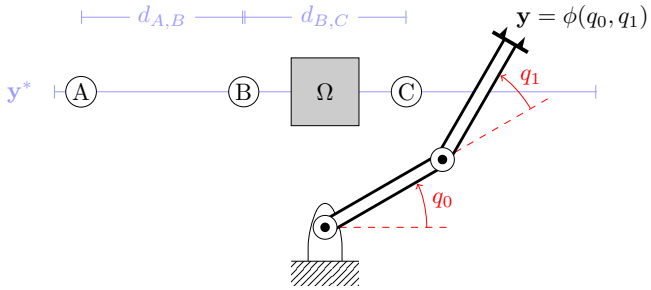

Fig. 3: Planar reaching task: Given a start state $\mathbf{x} = (q_0, q_1)$ (which maps to task space $\mathbf{y} = \phi(\mathbf{x})$), a robot has to reach a task space goal $\mathbf{y}^* = B$ while avoiding obstacle Ω. The warm-start solutions for task space goals A and C are equidistant in $\mathbf{y}^*$ from B. Thus, given the same $\mathbf{x}$, both trajectories appear to be equally suitable warm-starts to a simple distance metric.

by the maximum joint space velocity and original feasible joint space path length. In order to compare costs across variable length time horizons, we normalize the cost defined in Equation (1) over the duration of the trajectory. These initializations can be generated online (at query time) at the cost of computing a feasible plan from scratch. However, if a task (or a family of tasks) gets repeated, a similar or same query has to be computed over and over. An efficient solver would compute these solutions offline and store them in a library. We begin creating a library of motion by defining an indexing scheme and precomputation strategy.

A. Problem Encoding

A task in an environment Ω is fully described by the initial floating base placement and joint configuration at the start $\mathbf{x}_0$ and task space goals $\mathbf{y}^*$. The relationship between the task space $\mathbf{y}$ and environment Ω can hereby be captured and encoded through parametrized obstacles, topological [19] or geodesic coordinates [20], relative distances [18], or learned descriptors [13]. In this work, we focus on initialization strategies in a known environment leaving considerations regarding generalization across environments to future work. Start state $\mathbf{x}_0$, task space goals $\mathbf{y}^*$, and environment parametrization Ω together form a fully specified planning problem $(\mathbf{x}_0, \mathbf{y}^*, \Omega)$ which maps to an optimal trajectory $X^* = (\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_T)$ if it is solvable:

$$(\mathbf{x}_0, \mathbf{y}^*, \Omega) \mapsto (\mathbf{x}_0, \cdots, \mathbf{x}_T) \tag{6}$$

For a robot with a floating base, we capture the relative relation between the robot and the environment by expressing the encoding in an environment frame (e.g., a landmark). In this case we adjust the notation:

$$(\mathbf{x}_{0,\Omega}, \mathbf{y}^*, \Omega) \mapsto (\mathbf{x}_0, \cdots, \mathbf{x}_T), \tag{7}$$

where $\mathbf{x}_{0,\Omega}$ has the floating base frame expressed with respect to the environment frame.

As the problem encoding captures the relative aspect between the robot and an encoded environment, we can augment the dataset by re-using sub-trajectories of the original

Algorithm 1 Create Subindexing in Problem-Solution-Map

Require: Precomputed Library
Ensure: Augmented Library (with subindexing)
  for all Sample $\in$ Library **do**
    for $t = 1$ **to** $t = T - 2$ **do**
      $Problem = (\mathbf{x}_{t,\Omega}, \mathbf{y}^*, \Omega)$
      $X_{optimal} = (X_t, \cdots, X_T)$
      $Start = t$
      $Length = T - t$
      $StoreSubIndex(Problem, Start, Length)$
    for $t = T - 1$ **to** $t = 0$ **do**
      $\mathbf{y}_t^* = \phi(\mathbf{x}_t)$
      for $i = 0$ **to** $i = t - 2$ **do**
        $Problem = (T_{\text{Base},\Omega}, \mathbf{q}_i, \mathbf{y}_t^*, \Omega)$
        $Start = i$
        $Length = t - i$
        $StoreSubIndex(Problem, Start, Length)$

solutions as any part of an optimal trajectory can form a near-optimal solution for a similar problem. In particular, we consider two cases for subindexing:

1) The sub-trajectory starting from state $\mathbf{x}_{0<t<T} \in X$ to the original task space goal $\mathbf{y}^*$ forms a near-optimal solution for a problem described by $(\mathbf{x}_{t,\Omega}, \mathbf{y}^*, \Omega)$.
2) The sub-trajectory from state $\mathbf{x}_{0<t<T}$ to the task space goal defined by a subsequent state $\mathbf{y}_t^* = \phi(\mathbf{x}_{t<i<T})$.

Algorithm 1 shows the procedure for creating these subindices. Together with the goal region growing, this results in a problem-solution-map.

B. Trajectory Library Precomputation

It is important to note that the task space goals $\mathbf{y}^*$ as a parameter of (1) may change the landscape of the objective function significantly and discontinuously. A good warm-start seed should be local in the value function (have a similar cost landscape), however, this may not translate to closeness in the chosen problem encoding space. Consider the two-link reaching task depicted in Figure 3, where the end-effector has to repeatedly reach different positions $\mathbf{y}^*$ along the blue line. By solving the problem from varying start configurations $\mathbf{x}_0 = (q_0, q_1)$ for changing task space goals $\mathbf{y}^*$, we can create a library mapping problem encodings to optimal trajectories. A key consideration hereby is the trade-off between precomputation time and storage size and online retrieval, convergence time, and adaptation success rate. Exhaustively enumerating the problem encoding (and thus the value function) is intractable for higher-dimensional problems. Finding representative sample trajectories and determining a sufficient sample density is non-trivial as the cost landscape (value function) of the task is not known a priori.

Random sampling in the space of the problem encoding is unlikely to achieve sufficient coverage in task-relevant parts of the space, with many samples prone to being unused and possibly resulting in a low warm-start success rate. Deterministic sampling in a discretized problem encoding space

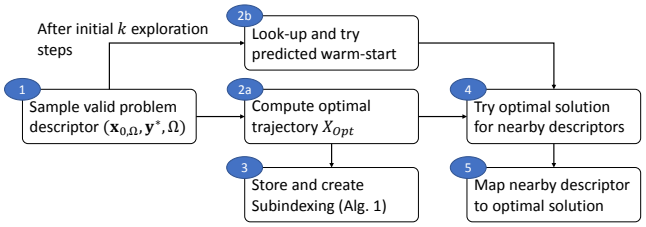

Fig. 4: The precomputation procedure for building the trajectory library: First, a valid problem descriptor is sampled in the space of the problem encoding and an optimal solution computed offline. The optimal trajectory is stored and subindexed using Algorithm 1. Finally, goal region growing is performed by testing the adaptability to nearby descriptors.

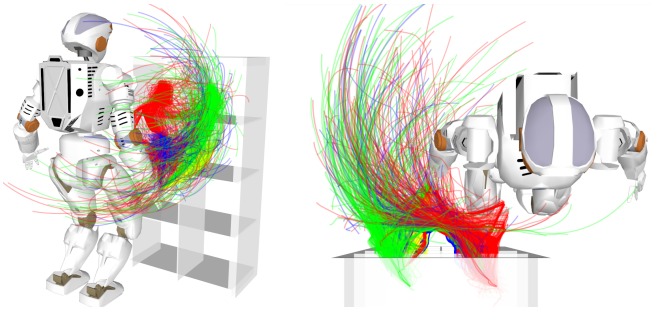

Fig. 5: Visualization of the original optimal trajectories in the library. The class labels (color) have been assigned based on closest task space goal $\mathbf{y}^*$.

can result in the issue highlighted in Figure 3: Depending on the resolution of the discretization, two warm-start seeds (A and C) may appear equidistant to a simple distance metric. However, this does not necessitate a similarity in the optimal solutions (e.g., mode changes introduced by the environment, dynamics, etc.). On the other hand, the Euclidean metric in the problem encoding space is fast to compute and a scaling of the distance metric can be used to improve the quality of the retrieved initial trajectories. Thus, with a focus on a suitable sampling bias and strategy, efficient look-up can be performed using inexpensive distance metrics.

Our precomputation and exploration strategy is depicted in Figure 4. In general, we aim for density in the problem encoding space while storing only a limited number of solutions as computing new optimal trajectories from scratch is expensive, while adapting existing solutions is comparatively cheap. As such, upon obtaining a new optimal solution, we attempt to explore its region of validity by sampling in the neighborhood of the problem encoding and running an adaptation optimization step with a given time and iteration budget (goal region growing). If the problem converges, we add a mapping from the new sampled problem descriptor to the original optimal solution. Thereby, we achieve density in the problem encoding for a small number of original optimal trajectories. This can be interpreted as creating a basin of attraction around the sample trajectory.

In our approach, we use a combination of task space goal seeds, random exploration, and validity region growing to build an encoded trajectory library mapping problem encoding descriptors to variable-length optimal trajectories. At query time, we predict the best warm-start seed given a problem encoding by classification.

V. Evaluation

Having precomputed a library of warm-start trajectories, we now benchmark the proposed warm-start method against zero motion, straight-line linear interpolation, and an online RRT-Connect-based feasible initialization. We also compare different nonlinear program solvers. For fair comparison, we use the same forward kinematics, collision checking, objective and Jacobian computation with the planning prototype and benchmarking library EXOTica [21]. Performance differences between solvers thus correspond to the number of evaluations each algorithm requires and their internal updates. While the problem formulation and solvers are implemented in C++, we use Python for high-level logic and warm-start look-up. All evaluations are carried out on a computer with an Intel Core i7-6700K CPU with $4\,\mathrm{GHz}$ base frequency and 32GB $2133\,\mathrm{MHz}$ memory in a single thread, with several independent benchmarks or precomputations running in parallel. All our evaluations use densely sampled time horizons with $\Delta t = 0.05s$.

A. Convergence Using Different Solvers

We evaluate the problem formalized in Section III with a probabilistic inference solver (AICO [8]), quasi-Newton methods (BFGS and L-BFGS), as well as conjugate gradient descent for a humanoid shelf picking task. We use our own implementation of AICO, the open-source quasi-Newton and nonlinear conjugate gradient solvers from OPT++ [22], as well as a commercial L-BFGS implementation (SNOPT [23]).

The cost evolution against time and iterations for optimization until convergence from a feasible, RRT-Connect initialization are shown in Figure 6. AICO is quick to make progress and achieves the lowest overall cost (as it updates individual states repeatedly without a full roll-out), however, it may return invalid trajectories (i.e., those which diverge from the final target position or have a sample in collision). Quasi-Newton methods do not achieve a similarly low cost as AICO as they stay within the originally provided local optimum. This, however, leads to a higher success rate from feasible initializations as it would not update individual states to be in collision in order to satisfy a smoother transition cost. As expected, full BFGS updates require more function and gradient evaluations to estimate the Hessian and are thus slower than their limited-memory variant. In turn, they are able to use the more accurate Hessian to achieve a lower final cost. This performance gain of L-BFGS is more pronounced for longer time horizons. Overall, as AICO and SNOPT have the highest convergence rates with the lowest costs, we will focus on them in the following benchmark experiments.

B. Pick-and-Place Benchmark

For the shelf benchmark, we compare different initialization strategies for use with AICO and SNOPT's L-BFGS: Common (zero-motion and straight-line interpolation between start and goal configurations), feasible (RRT-Connect), and the proposed trajectory library with problem encoding.

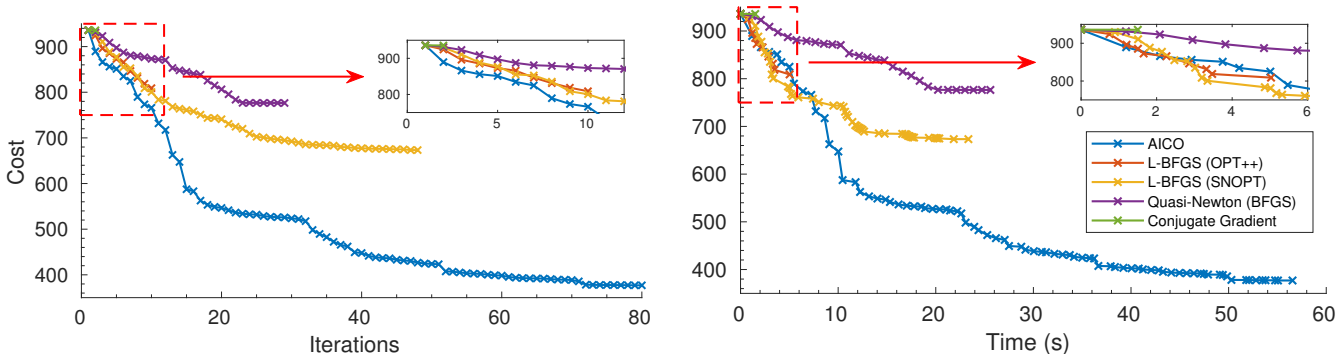

Fig. 6: Cost evolution for different optimization solvers using the same, feasible initialization seed. *Left:* Cost vs. iterations. *Right:* Cost vs. time. AICO initially improves the fastest and achieves the lowest overall cost. For quasi-Newton solvers, a commercial implementation with limited memory BFGS updates performs best (SNOPT, orange). For open source quasi-Newton solver implementations, BFGS achieves a lower cost than its limited memory variant (OPT++, red), however, at the cost of taking more time per step to estimate the full Hessian.

The trajectory library used in the benchmark contains $3,233$ original solutions with $36,256$ problem-encoding-to-solution-mappings after subindexing and goal region growing ($24\,\mathrm{MB}$, $12.6\,\mathrm{h}$ single thread precomputation time). These original trajectories are shown in Figure 5. For feasible initializations during offline precomputation, we use a bidirectional sampling-based planning algorithm (RRT-Connect [24]) with L-BFGS as the optimization and goal region growing/adaptation solver. We initialize the exploration with task space goals to reach into each shelf compartment from a nominal whole-body posture as well as from within each compartment to every other compartment. We apply goal region growing and limit the number of additional random exploration steps to 500.

For fast indexing, we compare k-Nearest Neighbor (KNN) with exhaustive search over the problem encoding as retrieval strategies. In particular, we consider the following settings:

1) KNN with an Euclidean distance metric tested both with and without scaling by normalizing on the range (rows 7–10). The normalization is helpful as the problem encoding can include metric with non-metric information unknown to the look-up distance metrics.
2) Exhaustive search by retrieving the solution of the problem descriptor with the minimum norm, i.e., the closest in the problem encoding. We compare normalized and unscaled problem encodings (rows 11–14) with a task-informed distance metric (rows 15–16).

For comparability, all time horizons in a single benchmark request are equal and determined by the length of the time-spaced and interpolated RRT-Connect solution. On runtime, we sample uniformly in the range of the problem encoding constraining the task space goal $\mathbf{y}^*$ to the task domain with added Gaussian noise and ensure that each problem is solvable through the existence of a feasible solution.

C. Analysis of Results

Averaged results over $1,000$ benchmark requests are presented in Table I. Based on these, we can draw a number of observations highlighting different properties of the tested initialization strategies:

a) As expected, feasible, RRT-Connect initialization leads to convergence in the majority of cases: 96.1% with L-BFGS (row 6) and 94.8% with AICO (row 5). However, this comes at a $40-80\%$ higher final cost as the feasible initialization does not consider the other objectives of the cost function beyond collision avoidance and reaching the final configuration. The flexibility of a guaranteed feasible solution thus comes at the cost of a longer warm-start time as a new solution is computed from scratch.

b) Zero motion (which consequently has a large distance to the task space goals) and straight line interpolation (which may be infeasible and pass through obstacles) rarely succeed for quasi-Newton methods which require to be initialized in the basin of attraction of a local minimum (11.6% and 14.8% success rates, respectively). Where they converge on a solution, the solutions have costs up between 4 and 40 times higher than the best-performing initialization strategy (row 15). This is expected as if in collision, quasi-Newton methods are trapped (in fact this is the source of all failures).

c) AICO manages to converge with 33.4% and 14.4% in these scenarios, although at an up to twice higher cost compared with being initialized with a feasible solution while requiring similar time or longer. When optimization fails, it is largely due to not achieving the final task space goal for zero motion (74.2%), and almost exclusively due to being stuck in collision for straight-line initialization (94.4%).

d) An exhaustive exploration of the trajectory library using a normalized problem encoding yields a 96.0% success rate along with the lowest overall final cost for an automatic distance metric suggesting the exploration strategy produced a library with good coverage on the task domain. Use of a task-informed, hand-tuned metric can boost success rates to 99.5% while reducing mean final cost by a further 16.2%. This suggests that future investigations on automatically learning distance metrics or generalizing classifiers can ensure a more efficient use of the generated trajectory library.

e) Among the KNN variants, the one trained on the normalized problem encoding performed the best—with similar success rates for both AICO and L-BFGS; rivaling the success rates of feasible initialization while obtaining a lower final cost with up to twice quicker convergence. Note, the results in rows 7–10 equal those in rows 11–14 as the nearest neighbor algorithm selected the same initial seeds as the exhaustive search, while being more efficient in the

	Solver	Initialization	Success Rate	Warm-Start Time (ms)	Residual Cost	Time to Convergence (s)	Iterations until Convergence
1	AICO	Zero-Motion	33.40%	–	1912.60 ± 2426.13	53.44 ± 56.03	139.32 ± 141.72
2	L-BFGS		11.60%		1729.23 ± 1209.04	34.09 ± 24.15	292.91 ± 166.14
3	AICO	Straight-Line	14.40%	–	1483.56 ± 3293.76	31.05 ± 36.02	129.82 ± 144.10
4	L-BFGS		14.80%		16923.00 ± 37558.22	17.64 ± 19.48	180.27 ± 210.52
5	AICO	RRT-Connect	94.80%	619.25 ± 359.07	927.83 ± 3203.49	34.62 ± 35.43	71.55 ± 70.62
6	L-BFGS		96.10%		909.81 ± 513.43	27.12 ± 24.99	166.01 ± 164.38
7	AICO	KNN	64.10%	0.87 ± 0.11	653.00 ± 550.46	34.70 ± 35.51	75.06 ± 73.68
8	L-BFGS		65.20%		866.54 ± 510.07	20.95 ± 17.65	120.63 ± 136.46
9	AICO	KNN (normalized problem encoding)	**96.00%**	1.18 ± 0.20	$\mathbf{496.51 \pm 306.24}$	19.87 ± 28.06	56.33 ± 63.18
10	L-BFGS		95.80%		667.50 ± 409.58	$\mathbf{14.17 \pm 15.18}$	152.49 ± 158.69
11	AICO	Exhaustive search	64.10%	3.72 ± 0.35	653.00 ± 550.46	34.60 ± 35.29	75.06 ± 73.68
12	L-BFGS		65.20%		866.54 ± 510.07	20.97 ± 17.76	120.63 ± 136.46
13	AICO	Exhaustive search (normalized problem encoding)	96.00%	3.73 ± 0.37	496.51 ± 306.24	19.89 ± 28.08	56.33 ± 63.18
14	L-BFGS		95.80%		667.50 ± 409.58	14.18 ± 15.21	152.49 ± 158.69
15	AICO	Exhaustive search (task-informed weights)	98.30%	3.71 ± 0.31	$\mathbf{416.02 \pm 286.37}$	35.37 ± 37.87	75.99 ± 69.20
16	L-BFGS		**99.50%**		741.85 ± 442.54	15.65 ± 20.44	110.64 ± 146.28
17	RRT-Connect	–	100.00%	–	5163.96 ± 26328.00	0.62 ± 0.36	–

TABLE I: Overview of success rate, convergence time and iterations for different optimization solvers and initialization strategies across 1,000 benchmark trials with initial states uniformly sampled within the valid range and task space goals sampled from a normal distribution centered at the shelf compartments. The zero-motion initialization is e.g. used by RieMO, while the straight-line initialization is used by CHOMP and STOMP. We use sampling-based initialization during our precomputation stage as a feasible initialization and investigate different look-up/indexing strategies based on our problem encoding. We ensure that all benchmarking scenarios are solvable by RRT-Connect. We use our own implementation of AICO and SNOPT's version of L-BFGS.

look-up due to its data structure. The success rate with L-BFGS is marginally lower compared to being initialized with RRT-Connect, however, achieves a lower cost twice as fast.

f) For our trajectory library, KNN is 3.1 times faster to look up a warm-start than searching the library exhaustively—this difference is expected to grow with increasing number of stored problem-solution-mappings. The KNN retrieval features the fastest warm-start time (within approximately $1.2\,\mathrm{ms}$), and either KNN or exhaustive search are at least two orders of magnitude faster than running SBP from scratch.

g) Normalizing the problem encoding to account for different scalings provides better warm-start seeds and higher convergence independent of the look-up method as it allows for simple distance metrics to be used successfully (49.8% and 46.9% increases in success rates, respectively).

Overall, the benchmark results suggest that given a known environment and the presented precomputation strategy, the proposed problem encoding can be efficiently used to warm-start trajectory optimization solvers online using nearest neighbor look-up with inexpensive distance metrics.

VI. Discussion

This paper considered the problem of providing a warm-start initialization for trajectory optimization algorithms in complex environments which result in highly nonlinear objective functions. We hereto proposed the use of an offline generated trajectory library with a problem encoding which can be used for fast online indexing based on inexpensive nearest neighbor metrics. We detailed strategies for efficient sample reuse and evaluated our method in a randomized benchmark on a shelf picking task. Our results demonstrate that our method achieves similar or higher success rates to initialization with feasible solutions from sampling-based planners while converging faster with lower final costs.

While we considered only the closest candidate from indexing and look-up based on similarity metrics, other work has focused on training generalizing classifiers for single or ordered list prediction. [25], e.g., used exhaustive online training after library generation to obtain a series of classifiers for list prediction allowing a sequential initialization with the next best candidate should previous warm-starts fail.

A key challenge for applying regression over warm-starts are discontinuities, e.g., introduced by dynamics or complex environments. While we focused on high-dimensional, kinematic planning with discontinuous objective functions introduced by complex environments, similar precomputation and initialization approaches have been presented, e.g., for dynamic lower-DoF optimal control problems in obstacle-free environments. Here, [14] presented an iterative approximation of the value and policy functions initialized from and used in building an optimal kinodynamic PRM. While the nature of the underlying problems differ, we consider iterative approximation *during* offline precomputation a highly interesting avenue for future work. Graph-based approaches, e.g., HDRM [17], PRM and its variant Experience Graphs [26] which are based on prior solutions with heuristics guiding search onto previously explored graphs, are interesting, but may limit exploration and are only optimal w.r.t. the roadmap.

One of the key limitations of our formulation is the use of soft constraints, e.g., for collision and joint limit avoidance as well as task space targets. While high penalties on these terms often lead to desired behavior, there are no

guarantees on satisfaction of these constraints (requiring a collision check of the resulting optimal trajectory) and different cost terms may act against each other causing early, sub-optimal, or no convergence. In practice, using unconstrained optimization is efficient and fast given a good initialization, e.g., from a trajectory library as considered in this work. In recent years, efficient solvers for constrained nonlinear programs have shown impressive results in literature, e.g. using SQP [1], [23]. In order to reduce failures from otherwise suitable initializations, future work should explicitly incorporate equality (e.g., reach targets), inequality (e.g., joint velocity limit and collision avoidance), and bound (e.g., joint limit) constraints. Furthermore, using analytic second-order information where available and exploring a staged cost function or adaptation of weights in an outer loop can address poor initialization and speed up convergence.

In our work, we use the signed distance between the robot and environment as a collision proxy without preprocessing. Preprocessing, e.g., convex decomposition [1], using signed distance fields and overlapping sphere approximation of the robot body itself [4], [6] could speed up collision queries which currently dominate the cost and prevent failures from discontinuities, e.g., when initialized in collision. Additionally, evaluating collisions only at discrete time steps requires a dense discretization of the trajectory while not guaranteeing continuous time safety (e.g., as considered in [1]). Further, dense discretization of the time horizon results in large optimization problems, with opportunities to speed up convergence by using trajectory parameterizations and embeddings such as Gaussian Processes [3], B-splines, dynamical systems, kernel methods [20], [27], [28], or subsampling of an initial coarse time parametrisation [29].

As for the trajectory library, defining the task space encoding and distance metric for efficient look-up is important. Avenues for future work include clustering of solutions into distinct classes (e.g., from topology), learning of feature descriptors (in lieu of the engineered task space encoding) and/or distance metrics, as well as investigating the use of function approximation for value and policy functions. Here, learning disentangled policy predictions that regress within distinct solution classes are particularly interesting. Further, the current formulation only handles static obstacles and warm-starts in dynamic scenes remain an open question.

Additionally, as discretized or exhaustive sampling is prohibitive, future work should address exploration strategies for the offline dataset generation stage when posed with more complex problems. Beyond exploration, limitations on memory and online look-up runtime motivate information-theoretical considerations on how many samples are required and which ones can be discarded.

Finally, this work considered high-dimensional problems in complex environments with implicit first-order Markov dynamics. It is an interesting area for further investigation to extend it to include tasks with complex dynamics such as switching contacts or underactuated systems as the optimization problems become more expensive to evaluate and thus a near-optimal warm-start more effective.

REFERENCES

[1] J. Schulman, J. Ho, *et al.*, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *R:SS*, 2013.

[2] M. Toussaint, "A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference," in *Geometric and Numerical Foundations of Movements*, J.-P. Laumond, Ed. Springer, 2017.

[3] J. Dong, M. Mukadam, *et al.*, "Motion planning as probabilistic inference using gaussian processes and factor graphs." in *R:SS*, 2016.

[4] N. Ratliff, M. Zucker, *et al.*, "CHOMP: Gradient optimization techniques for efficient motion planning," in *IEEE ICRA*, 2009.

[5] N. Ratliff, M. Toussaint, and S. Schaal, "Understanding the geometry of workspace obstacles in motion optimization," in *IEEE ICRA*, 2015.

[6] M. Kalakrishnan, S. Chitta, *et al.*, "STOMP: Stochastic trajectory optimization for motion planning," in *IEEE ICRA*, 2011.

[7] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *Journal of Machine Learning Research*, vol. 11, no. Nov, pp. 3137–3181, 2010.

[8] M. Toussaint, "Robot trajectory optimization using approximate inference," in *ACM ICML*, 2009.

[9] D. Pavlichenko and S. Behnke, "Efficient stochastic multicriteria arm trajectory optimization," in *IEEE IROS*, 2017.

[10] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental Trajectory Optimization for Real-Time Replanning in Dynamic Environments." in *ICAPS*, 2012.

[11] C. G. Atkeson and J. Morimoto, "Nonparametric representation of policies and value functions: A trajectory-based approach," in *NIPS*, 2003.

[12] M. Stolle and C. G. Atkeson, "Policies based on trajectory libraries," in *IEEE ICRA*, 2006.

[13] N. Jetchev and M. Toussaint, "Fast motion planning from experience: trajectory prediction for speeding up movement generation," *Autonomous Robots*, vol. 34, no. 1-2, pp. 111–127, 2013.

[14] N. Mansard, A. Del Prete, *et al.*, "Using a Memory of Motion to Efficiently Warm-Start a Nonlinear Predictive Controller," in *IEEE ICRA*, 2018.

[15] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[16] Y. Yang, V. Ivan, *et al.*, "Scaling sampling-based motion planning to humanoid robots," in *IEEE ROBIO*, 2016.

[17] Y. Yang, W. Merkt, *et al.*, "HDRM: A Resolution Complete Dynamic Roadmap for Real-Time Motion Planning in Complex Scenes," *IEEE RA-L*, vol. 3, no. 1, Jan 2018.

[18] Y. Yang, V. Ivan, and S. Vijayakumar, "Real-time motion adaptation using relative distance space representation," *IEEE ICAR*, 2015.

[19] V. Ivan, D. Zarubin, *et al.*, "Topology-based representations for motion planning and generalization in dynamic environments with interactions," *IJRR*, vol. 32, no. 9-10, pp. 1151–1163, 2013.

[20] M. Sugiyama, H. Hachiya, *et al.*, "Geodesic gaussian kernels for value function approximation," *Autonomous Robots*, vol. 25, no. 3, pp. 287–304, Oct 2008.

[21] V. Ivan, Y. Yang, *et al.*, "EXOTica: An Extensible Optimization Toolset for Prototyping and Benchmarking Motion Planning and Control," in *Robot Operating System (ROS): The Complete Reference (Volume 3)*, A. Koubaa, Ed. Springer, 2019.

[22] J. C. Meza, R. A. Oliva, *et al.*, "OPT++: An object-oriented toolkit for nonlinear optimization," *ACM Trans. on Math. Softw. (TOMS)*, 2007.

[23] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Rev.*, 2005.

[24] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *IEEE ICRA*, 2000.

[25] D. Dey, T. Y. Liu, *et al.*, "Contextual sequence prediction with application to control library optimization," in *R:SS*, 2013.

[26] M. Phillips, B. J. Cohen, *et al.*, "E-graphs: Bootstrapping planning with experience graphs." in *R:SS*, 2012.

[27] K. Rawlik, M. Toussaint, and S. Vijayakumar, "Path Integral Control by Reproducing Kernel Hilbert Space Embedding." in *IJCAI*, 2013.

[28] Z. Marinho, A. Dragan, *et al.*, "Functional gradient motion planning in reproducing kernel hilbert spaces," in *R:SS*, 2016.

[29] D. Mitrovic, S. Klanke, and S. Vijayakumar, "Adaptive optimal feedback control with learned internal dynamics models," in *From Motor Learning to Interaction Learning in Robots*, O. Sigaud and J. Peters, Eds. Springer, 2010, pp. 65–84.