Journal Title
XX(X):1–23
©The Author(s) 2021
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Motion planning in task space with Gromov-Hausdorff approximations

Fouad Sukkar, Jennifer Wakulicz, Ki Myung Brian Lee, and Robert Fitch

Abstract
Applications of industrial robotic manipulators such as cobots can require efficient online motion planning in environments that have a combination of static and non-static obstacles. Existing general purpose planning methods often produce poor quality solutions when available computation time is restricted, or fail to produce a solution entirely. We propose a new motion planning framework designed to operate in a user-defined task space, as opposed to the robot's workspace, that intentionally trades off workspace generality for planning and execution time efficiency. Our framework automatically constructs trajectory libraries that are queried online, similar to previous methods that exploit offline computation. Importantly, our method also offers bounded suboptimality guarantees on trajectory length. The key idea is to establish approximate isometries known as ϵ-Gromov-Hausdorff approximations such that points that are close by in task space are also close in configuration space. These bounding relations further imply that trajectories can be smoothly concatenated, which enables our framework to address batch-query scenarios where the objective is to find a minimum length sequence of trajectories that visit an unordered set of goals. We evaluate our framework in simulation with several kinematic configurations, including a manipulator mounted to a mobile base. Results demonstrate that our method achieves feasible real-time performance for practical applications and suggest interesting opportunities for extending its capabilities.

Keywords
Task-Space Planning, Gromov-Hausdorff Approximations, Path Planning for Manipulators, Motion Planning, Task Sequencing

1 Introduction

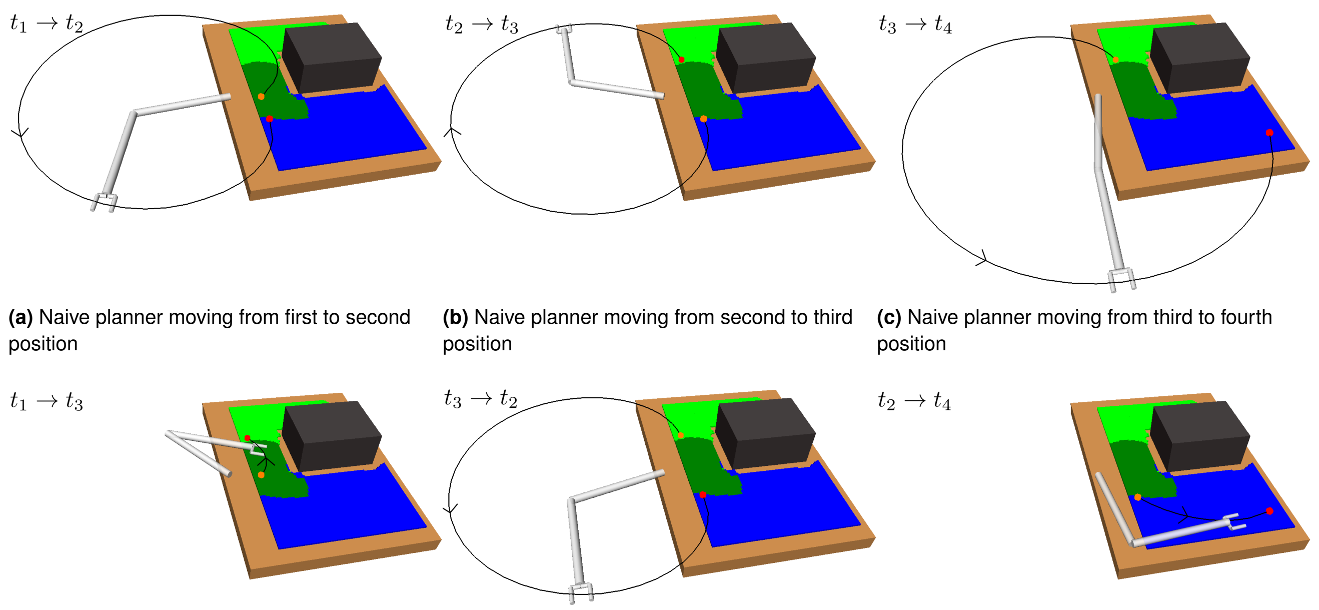
Motion planning for emerging applications of robotic manipulators must support a greater degree of autonomy than has traditionally been necessary. Industrial robotic manipulators such as *cobots* (collaborative robots), for example, are designed for advanced manufacturing applications where they should operate safely in dynamic work environments shared with humans and should be able to adapt quickly to perform a variety of tasks. These applications share a need for agility and rapid deployment that differs substantially from traditional applications of industrial manipulators, which are typically characterised by repetitive motions in highly structured environments with planning performed offline. Although existing motion planning algorithms have desirable computational properties such as probabilistic completeness, they are often limited in their ability to reliably produce feasible trajectories online in high-tempo tasks in practice.

We are interested in developing efficient algorithms for certain practical situations that require repeated, rapid, and reliable planning, including cobots and advanced manufacturing applications. A common example is a manipulator that must grasp objects from shelves and cabinets in a domestic or warehouse environment (Srinivasa et al. 2012; Dogar et al. 2013; Morrison et al. 2018). The shelves are static and their dimensions can be measured beforehand; however, the objects on the shelves might not be known and their locations can change. More precisely, we consider operational scenarios where a series of motion plans with differing goal configurations must be computed for a given environment. The environment consists of static elements such as fixtures and equipment plus potentially unknown objects that must be perceived online.

Classical approaches such as the probabilistic road map (PRM) (Kavraki et al. 1996) aim to gain efficiency through computational reuse; a computationally costly offline process generates a data structure that can then be repeatedly queried efficiently by a low-cost online process. However, this approach can become unwieldy in practice for robotic manipulators, which often have six or more degrees of freedom (DOF) and a configuration space of equivalent dimension. It is necessary to resolve the tradeoff between the size of the precomputed data structure, which grows naively with the complexity of the manipulator kinematics and environment, and the speed of online queries, whose complexity grows proportionally. Kinematic redundancy of robotic manipulators also introduces ambiguity in goal configuration when goals are specified only in terms of end-effector pose.

Authors are with the School of Mechanical and Mechatronic Engineering, University of Technology Sydney, Ultimo, NSW 2006, Australia

Corresponding author:
Fouad Sukkar, University of Technology Sydney, 15 Broadway, Ultimo, NSW 2007, Australia.
Email: Fouad.Sukkar@uts.edu.au



(a) Naive planner moving from first to second position **(b)** Naive planner moving from second to third position **(c)** Naive planner moving from third to fourth position

(d) HAP moving from first to third position **(e)** HAP moving from third to second position **(f)** HAP moving from second to fourth position

Figure 1. A two-DOF robotic manipulator tasked with moving its end effector to four unordered positions on a table-top environment with a box obstacle (dark grey). Positions are shown as coloured dots. The end-effector trajectory is drawn with direction of movement indicated by the arrowhead. Green and blue regions are subspaces within which end-effector motion does not require large changes in configuration. Overlap is shown in dark green. (a)-(c) show a sequence of trajectories produced by a naive planner. (d)-(f) show the sequence produced by our HAP method. HAP's choice of sequencing exploits short within-subspace trajectories in (d) and (f), whereas the naive planner's choice of initial position and configuration results in long trajectories to satisfy kinematic constraints.

One way to limit the expanse of roadmap-like data structures, and thus improve query time, is through the notion of *task space*. Unlike configuration space and work space, which are defined by the physical design of the robot system, task space is defined by the user according to the tasks at hand (Yu 2018). An intuitive example is to define a planar task space for operations on a flat surface such as sanding or polishing. For pick-and-place operations, the task space can be naturally expressed as the set of end-effector poses that correspond to possible object locations. A set of trajectories can be generated that correspond to common start-goal locations, for example, and reused online (Ellekilde and Petersen 2013; Lee et al. 2014). Unfortunately, this approach requires substantial effort from the user since the set of precomputed trajectories must be designed for each set of tasks. It would be preferable to automate this process to allow the robot system to be redeployed quickly for new tasks and changes in the configuration of the environment, and to design the library of trajectories in a way that facilitates predictable execution online over long task horizons.

A desirable property of motion planning in task space is that, for any two tasks close to each other in task space, one expects a smooth, short trajectory to exist between them. The key idea we propose in this paper is to construct trajectory libraries that satisfy this property by establishing a bounding relation between distances in task space and corresponding distances in configuration space, defined as a mapping between metric spaces known as an ϵ-Gromov-Hausdorff approximation (ϵ-GHA). The aim is to ensure that a short path in task space implies a short path in configuration space, which additionally encourages smoothness.

We present the *Hausdorff approximation planner (HAP)* based on the key idea of constructing trajectory libraries with ϵ-GHAs. HAP builds a set of precomputed trajectories that resemble roadmap methods in that trajectories can be concatenated to satisfy a sequence of queries as described in our motivating scenario. To allow concatenation, the trajectories are indexed by a set of maps from the task space (end-effector poses) to the configuration space (joint angles) that disambiguate kinematic redundancy. The maps are designed to be ϵ-GHAs, which are approximate isometries that, as we show, ensure that the trajectories are efficient in combination. The task space is thus divided into smaller subspaces of similar manipulator configurations where, for any two poses close to each other in a given subspace, there exists a short trajectory between resulting configurations according to the map. As a result, each trajectory is efficient individually as well as in combination.

Given a user-defined task space, HAP automatically computes a subspace decomposition and generates a set of trajectories that span it. These trajectories are stored in a library that can be modified to adapt to changes in the environment or to the task-space definition. A simple illustration of two overlapping task subspaces that map to two disjoint configuration subspaces for a 2-DOF manipulator is shown in Fig. 1. As can be seen in the figure, a naive approach results in unnecessary wasted motion as opposed to our method which utilises the subspace knowledge. In higher dimensions, subspaces for 6-DOF or 7-DOF arms might differ in shoulder-in versus shoulder-out configurations, for example. Because only the task space itself is user-defined, this aspect of the algorithm's design facilitates rapid deployment in practice.

We evaluate the behaviour of HAP in a single-query setting and then in a batch-query setting that requires it to find a minimum concatenated path through a set of unordered end-effector poses. The batch-query model could be used to take observations from a set of viewpoints with an eye-in-hand sensor, for example. To highlight HAP's applicability to a variety of manipulator platforms and task-space definitions, we perform experiments in simulation with 6-DOF and 7-DOF manipulators, and with a manipulator mounted to a mobile base. Planning time performance is favourable overall, which shows that the HAP algorithm is effective in choosing precomputed trajectories for rapid online planning even in the presence of physical constraints, such as problematic singularities and joint limits, that otherwise would lead to unnecessarily long or awkward trajectories. Single-query results show a decrease in planning time of at least 50% versus baseline methods. Batch query results show a similar decrease in planning time and also a near perfect success rate with low jerk, whereas baseline methods failed to find a plan within the allocated time budget in at least 10% of problem instances.

The main significance of this work is to improve the practical utility of autonomous robotic manipulators, particularly for non-experts. Automatically generating smooth, short trajectories online directly from a user-defined task space with reasonable performance guarantees is a step beyond current general purpose motion planning methods, which generally require expertise to design implementations that perform well in practice. Our HAP implementation is made available as supplementary material.

2 Related work

2.1 Trajectory libraries

A variety of methods have been proposed that precompute a library of trajectories and adapt them online. For example, Lin and Berenson (2016) utilise a trajectory library alongside a discrete planner for planning complex humanoid motions with palm contacts. Combining a trajectory library with a discrete planner is shown to improve planning success, particularly if the search space has a high branching factor.

A similar idea was explored by Lee et al. (2014) and Ellekilde and Petersen (2013) for robotic manipulators where a library of trajectories is precomputed over a roadmap of representative end-effector poses. These methods achieve lower online planning times, but exhibit inconsistent success rates and trajectory costs. This is partly because the online adaption does not consider collision avoidance. More importantly, these methods tend to ignore trajectory cost optimisation during library construction, which can lead to arbitrary performance when adapted online.

Our previous work (Sukkar et al. 2019; You et al. 2020) introduced a planning framework that builds on the ideas of Lee et al. (2014) and Ellekilde and Petersen (2013) and allows for non-static obstacles when adapting the library trajectories online. Library construction was designed to cover the entire task space with a single subspace, however, and we observed that this led to inconsistent performance that varied based on manipulator kinematics. Experiments with non-redundant 6-DOF manipulators were markedly less successful than with redundant 7-DOF manipulators, for example. We showed that coverage of the task space with a single subspace mapping was not possible and resulted in poorly biased trajectories for tasks outside of the mapped subspace.

The HAP framework we propose here resolves observed limitations of trajectory library methods by introducing the notion of organising task space as a set of multiple subspaces, and by providing bounded suboptimality guarantees through ϵ-GHAs. In doing so, it allows for consistent performance across a variety of manipulator types, including non-redundant systems and mobile bases.

2.2 Trajectory optimisation

Trajectory optimisation generally refers to the process of making incremental changes to an initial trajectory until an optimal or near-optimal solution is found. The initial seed trajectory can be chosen randomly or heuristically. Existing methods can generate trajectories that avoid obstacles while maintaining smoothness (Kalakrishnan et al. 2011; Park et al. 2012; Zucker et al. 2013b; Schulman et al. 2014; Mukadam et al. 2018), and have been shown to be computationally efficient for high-dimensional systems (Toussaint 2009; Zucker et al. 2013a; Feng et al. 2015).

The major limitation of trajectory optimisation methods is susceptibility to local minima, which leads to sensitivity to choice of initial trajectory. Initialisation has been shown to have a significant impact on convergence speed and success rate, and there is no guarantee of finding a collision-free solution as the collision avoidance constraints in the optimisation problem are non-convex (Schulman et al. 2014).

A simple initialisation strategy is to draw a straight-line path between start and goal configurations in configuration space, ignoring obstacles. Others include attempts to learn classifiers that predict the effectiveness of random perturbations of a given seed trajectory (Tallavajhula et al. 2016; Pan et al. 2014; Dey et al. 2013). A recent learning-based approach (Natarajan 2021) uses long short-term memory (LSTM) neural networks, and exhibits better performance than straight-line initialisation.

The trajectory libraries constructed by HAP can be viewed as sets of high-quality seed trajectories that are precomputed based on manipulator kinematics and *a priori* knowledge of the environment. HAP uses trajectory optimisers to adapt these trajectories online.

2.3 Motion planning

General motion planning algorithms are the topic of decades of research. Good overviews can be found in textbooks (LaValle 2006; Choset 2005) and surveys (Elbanhawi and Simic 2014; Gammell and Strub 2020).

Sampling-based planners choose sample points from configuration space and connect them to build paths (Kuffner and LaValle 2000; Kavraki et al. 1996). Many variants have been proposed with a focus on probabilistic completeness and asymptotic optimality properties (Karaman and Frazzoli 2011; Arslan and Tsiotras 2013; Gammell et al. 2014; Janson et al. 2015). Batch Informed Trees (BIT*) (Gammell et al. 2015; Strub and Gammell 2020) is a sampling-based algorithm that uses incremental search techniques to incorporate new samples. BIT* has been shown to perform well, in particular, for high-dimensional problems.

Recently, Luna et al. (2020) proposed the XXL planner that computes high-quality trajectories for high-dimensional robots. XXL suffers from similar limitations as other sampling-based methods, however, and is probabilistically complete but offers no optimality guarantees.

Approaches for motion planning in task space, as we explore in this paper, are less common. Shkolnik and Tedrake (2009) proposed the TS-RRT algorithm, which is a modification of the RRT algorithm where samples are drawn from task space but connected in configuration space using techniques from task-space control. A benefit of this idea is that it avoids some of the challenges of sampling-based planning in high-dimensional configuration space, since task space is typically of lower dimension. For manipulators, this can also avoid the challenges of kinematic redundancy since paths in task space defined by end-effector pose are unique.

In order to satisfy the robot's kinematic constraints, however, a path must still be found in configuration space. Short paths in task space are not necessarily so in configuration space, and so the challenge remains of how to mitigate the potentially large changes in joint configurations when attempting to connect the task-space samples. The work we present addresses this issue by establishing ϵ-GHAs between the two spaces.

2.4 Task and motion planning

Task and motion planning (TAMP) problems extend motion planning to include high-level symbolic goals in a discrete action space (Driess et al. 2020) and thus are different in character from the problems we address here. However, TAMP methods are related in their consideration of sequences of motion planning tasks. This consideration arises because a sequence of actions in the discrete layer must map to a sequence of motion plans in configuration space. Most TAMP approaches focus on feasibility of the task sequence as opposed to efficiency of the resulting concatenated trajectories (Hauser and Latombe 2009; Srivastava et al. 2014; Toussaint 2015; Garrett et al. 2018).

Work that considers properties of concatenated trajectories includes methods for task sequencing (Alatartsev et al. 2013; Kovács 2013; Kolakowska et al. 2014; Kovács 2016) and multi-goal path planning (Wurll and Henrich 2001; Faigl et al. 2011). Related methods often formulate the problem as a variant of the well-known travelling salesman problem (TSP) (Alatartsev et al. 2015). Unfortunately, these approaches are generally too computationally expensive for online use and require full knowledge in order for plans to be computed offline.

For online planning, it can be beneficial to use heuristics that accurately estimate true trajectory costs yet are efficient to compute within the process of solving the underlying TSP. Finding such a heuristic is difficult, however, without incurring computational cost equivalent to solving the original motion planning problem (Hauer and Tsiotras 2017).

Heuristics for manipulators typically approximate trajectory costs by using Euclidean distance between start and goal points in the workspace (Silwal et al. 2017; Bac et al. 2017). This type of approximation can dramatically underestimate the true cost of the executed trajectory due to the nonlinearity of high-dimensional manipulators (Sukkar 2018), especially in environments with obstacles.

The trajectory libraries proposed here provide accurate costs that take into consideration the kinematic redundancy and nonlinear motion model of high-dimensional manipulators, and allow trajectories to be smoothly concatenated. Therefore, they can be used as efficient heuristics for task sequencing, as we demonstrate.

3 Problem formulation and approach

3.1 Notation

$\mathcal{C}$ represents the configuration space of the arm. The workspace, W, is the 3D Euclidean workspace, $W = \mathbb{R}^3$. Given a configuration $q \in \mathcal{C}$, $A(q) \subset W$ denotes the space occupied by the robot model at configuration q. $m \subset \mathbb{R}^3$ is an approximate model of the environmental obstacles. We assume access to a collision checking process that reports whether the arm is in collision with m or with itself. The obstacle region is defined as $\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} \mid A(q) \cap m \neq \emptyset\}$, from which we obtain the free space region $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$. A task t typically has a set of inverse kinematics (IK) solutions $\mathcal{Q}(t)$. Then, the task space $\mathcal{T} \subset SE(3)$ is the set of poses of the robot's end effector for which *valid* IK solutions, $Q(t) \subset \mathcal{Q}(t)$ exist. An IK solution q is considered valid if $A(q) \in \mathcal{C}_{\text{free}}$. $\hat{T}$ is a discrete approximation of the subset of $\mathcal{T}$ where the robot is expected to operate frequently.

3.2 Motion planning in task space: single-query problems

The manipulator is given a task modelled as a 6D pose $t \subset \mathcal{T}$ chosen from the task space. To complete the task, the manipulator must position its end effector at pose t while avoiding collision. The environment is not entirely known in advance, but a general model m is available that approximates what is expected. For example, m might represent a general bookshelf structure including the shelves and case (see Fig. 2). However, m need not include all objects within the bookshelf, as these details may be unknown *a priori* and discovered later.

The manipulator's trajectory is modelled as a discrete sequence of configurations $\pi = \{\pi[1], \ldots, \pi[N] \mid \pi[i] \in \mathcal{C}_{\text{free}}\}$. We measure the length of a trajectory using a metric on the configuration space,

$$d_C(\pi) = \sum_{n=1}^{N-1} d_C(\pi[n], \pi[n+1]). \tag{1}$$

We are interested in finding a minimum cost trajectory in $\mathcal{C}_{\text{free}}$ that completes task t. It is convenient to consider the starting pose of the end effector to be the goal pose of the previous task. We are therefore interested in the set of trajectories between task t_j and t_l, $\Pi(t_j, t_l) = \{\pi \mid \pi[1] \in Q(t_j), \pi[N] \in Q(t_l)\}$, leading to the following problem definition.

Problem 1. Motion planning in task space [single query]. *Find the shortest collision-free path π^* in configuration space between two tasks t_j and t_l in $\mathcal{T}$ such that*

$$\pi^*(t_j, t_l) = \underset{\pi \in \Pi(t_j, t_l)}{\arg\min}\ d_C(\pi). \tag{2}$$

3.3 Motion planning with multiple tasks: batch-query problems

The operational scenarios that motivate this work often involve more than one task. The manipulator is given an unordered set of M tasks $T = \{t_i\}_{i=1}^{M} \subset \mathcal{T}$. This set of tasks can be viewed as a batch-query scenario where all tasks must be achieved while minimising total cost. Thus it is necessary to choose a sequence of tasks, imposing a total ordering over T, in addition to repeatedly solving Problem 1. The batch-query problem can thus be formulated as follows.

Problem 2. Motion planning in task space [batch query]. *Find a permutation of the tasks $\sigma \in S_M$ that minimizes the total trajectory cost:*

$$\min_{\sigma \in S_M} \sum_{n=1}^{M-1} d_{\mathcal{C}}(\pi^*(t_{\sigma[n]}, t_{\sigma[n+1]})), \tag{3}$$

such that the configuration at the end of $\pi^*(t_{\sigma[n]}, t_{\sigma[n+1]})$ and the start of $\pi^*(t_{\sigma[n+1]}, t_{\sigma[n+2]})$ are equal for all $n \in [1, M-2]$.

It is challenging to approach this case directly. For each task pose $t_j \in T$ to be visited, there can be multiple valid configurations in set $Q(t_j)$. A direct solution requires simultaneously choosing the optimal configuration from $Q(t_j)$ and sequencing the tasks. In other words, this case contains an instance of set-TSP, which is more difficult than the standard TSP (Noon and Bean 1993) and potentially too computationally expensive for real-time planning since the number of possible sequences is $\mathcal{O}(|Q(t_j)|^M \times M!)$, where $|Q(t_j)|$ is the cardinality of the largest set of IK solutions for any task $t_j \in T$.

Mapping each task t_j to a single IK solution $q_{t_j} \in Q(t_j)$ transforms the set-TSP to a classical TSP, which is easier to solve. However, intelligent selection of a single IK solution from the possibly infinite set is also challenging.

Ideally, this transformation should accommodate efficient solution of Problem 1 with smooth, short trajectories between tasks. Therefore, we assume the availability of a mapping from each task t_j to a suitable unique IK solution $\theta(t_j) \in Q(t_j)$. Finding such a mapping thus arises as an important subproblem in our approach to solving Problem 2 and is defined as follows.

Problem 2.1. Mapping from task space to configuration space. *Find map $\theta : \hat{T} \to \mathcal{C}$ that assigns each task a unique IK solution that guarantees paths between tasks are short, smooth and collision-free in both task and configuration space.*

The kinematics of a robotic manipulator working in the task space may not necessarily admit a single map θ with the guarantee above. For example, in Fig. 1a there is no single θ that allows the arm to travel the short task-space distance required without taking a long path through configuration space. To handle such cases, we propose a further subproblem crucial to solving the problems presented above.

Problem 2.2. Finding regions of short travel in both task and configuration space. *Find K subspaces $\{\hat{T}^i\}_{i=1}^{K}$ of task space such that within each subspace a map $\theta^i : \hat{T}^i \to \mathcal{C}^i$ that satisfies the requirements in Problem 2.1 can be found.*

3.4 Approach overview

To ensure that the map $\theta : \hat{T} \to \mathcal{C}$ solves Problem 2.1, it is chosen to be an approximate isometry. Intuitively, an approximate isometry enforces that two tasks close in task space remain close in configuration space after mapping. Where a single approximate isometry does not exist, Problem 2.2 is solved by finding K approximate isometries. This decomposes the task space into subspaces that may overlap, leading to cases where a single task is assigned multiple IK solutions. Then, to solve Problem 2 we find solutions within each subspace independently after disambiguating which IK solution to use in overlapping subspace regions.

We construct the map(s) $\{\theta\}_{i=1}^{K}$ offline during the process of building a library of pre-computed trajectories tailored to the given task space. This library is made available to online algorithms designed to solve both types of query problems. Together, the offline and online algorithms form the HAP algorithmic framework.

4 Online motion planning with task subspaces

In this section, we present online algorithms to solve Problems 1 and 2. We first describe methods to retrieve, modify, and adapt trajectories drawn from the library in the single-query case. Then, we describe several additional steps necessary to address the batch-query case through task sequencing.

4.1 Trajectory retrieval and adaptation for single-query problems

Given a task drawn from the task space T, we must first identify matching tasks in the trajectory library constructed from a discrete set of anticipatory tasks $\hat{T}$ and retrieve corresponding trajectories. Examples of $\hat{T}$ and online tasks T are shown in Fig. 2. A retrieved trajectory is then modified such that its start and endpoints coincide with the current end-effector pose and that of the given task. These processes are summarised in Alg. 1 and detailed below.

4.1.1 Trajectory library matching Here we describe the library retrieval process where a single map θ is sufficient to solve Problem 2. Additional steps taken for the multiple map case are given in the next subsection. The trajectory library can be queried to retrieve a trajectory that connects a given pair of tasks drawn from the task space. We refer to endpoints of all such trajectories as *library tasks*, i.e., tasks stored within the library. Throughout this work, we assume that the task space is defined over the set of end-effector

Algorithm 1 Online Planner: Single Query

```
1: Given an online environment and task, retrieve matching
   library trajectories (Sec. 4.1.1)
2: Disambiguate IK solution (Sec. 4.1.2)
3: Augment chosen trajectory by appending start and goal
   configurations (Sec. 4.1.3)
4: Adapt trajectories to online environment (Sec. 4.2)
```

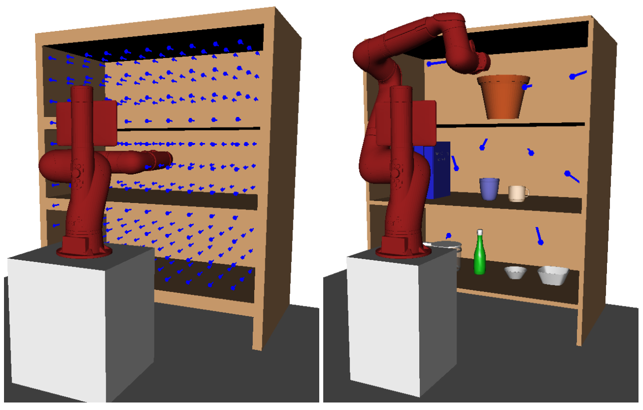


Figure 2. Example environment and task space next to an anticipated online scenario: (a) shows the discretised task space $\hat{T}$ (blue poses) and inflated environment m, (b) is an example online scenario with the arm in its home configuration. Online, HAP is robust against objects in (b) that are unmodelled in m and tasks can differ to those in $\hat{T}$.

poses and thus the term *task* can be assumed to refer to one such pose.

The library allows for retrieval of trajectories whose start and endpoints lie close to a given pair of online tasks. We now describe the process of evaluating such trajectories to choose the best match.

Given an online task t, we query the library to find a set of library tasks near t. We select the k-closest $\{\hat{t}_n\}_{n=1}^{k} \subset \hat{T}$ to t according to *task-space distance*. The task-space distance metric is user defined; here we use Euclidean distance between position components. To facilitate search, the library builds a KD-tree over task positions using this metric.

For the set of k-closest candidates, we retrieve their IK solutions assigned by θ, $\{\theta(\hat{t}_n)\}_{n=1}^{k}$, and compare them pairwise to all possible IK solutions $q_t \in Q(t)$ for task t using a suitable similarity metric in configuration space. We use the L_2-norm in this work. The pair of online/library task configurations $(q_t^*, \theta(\hat{t}_*))$ that minimises this metric is selected as a match.

HAP's matching process is different from previous work in that it depends on both task and configuration-space distances, rather than configuration-space distance alone (Lee et al. 2014; Ellekilde and Petersen 2013). Whereas configuration-space distance favours IK solutions that result in the least joint movement, task-space distance restricts the matching process to reflect the workspace geometry. To elucidate this claim, consider the bookshelf environment in Fig. 2. If matching is performed to minimise configuration-space distance only, an online task located in the middle shelf may be matched with a library task in the top shelf due to the small configuration change between them. Then, appending the online task to any retrieved trajectory corresponding to the top shelf library task would result in collision. On the other hand, if matching is performed to minimise task-space distance only, a matched library task is more likely to lie in the middle section of the bookshelf since shelf geometry is taken into account yet may result in inefficient, jerky trajectories due to potentially large differences in configuration space.

4.1.2 Subspace assignment During construction of the library, all tasks $\hat{t} \in \hat{T}$ are mapped to an IK solution by finding approximate isometry or isometries $\{\theta\}_{i=1}^{K}$. It is possible for these mappings to overlap, resulting in multiple IK solutions for a given task (one per map). Examples of the benefit of multiple mappings are given in Appendix B.1.

To handle the case where a library query results in multiple IK solutions due to overlapping mappings, it is necessary to define a process to choose one of them. It suffices to compare candidate IK solutions to the current robot configuration using the similarity metric discussed above (e.g., the L_2-norm in configuration space) and select the minimum. For computational efficiency, we select the first candidate whose similarity value falls below a given threshold.

The final special case to consider is one where the start and goal poses lie in disjoint mappings, and thus no trajectory exists in the library that connects them. To handle this case, a *home* configuration for the robot is defined in the library. The home configuration is chosen by the library construction procedure such that all subspaces may be reached via a straight-line trajectory in configuration space. Thus the home configuration acts as an intermediate point connecting all pairs of subspaces, and can be used to connect start and goal poses in disjoint mappings by simply joining the straight line trajectories to and from it.

4.1.3 Trajectory modification The matching trajectory chosen from the library will not align exactly with the start and goal poses of the robot in practical use. In general it is necessary for the library to be compact relative to task space in order to limit required storage to an acceptable level.

For start and goal poses (t_j, t_l), the corresponding IK solutions $q_{t_j}^*$ and $q_{t_l}^*$ are appended to the ends of the retrieved library trajectory. This results in a modified trajectory $\pi^*(t_j, t_l)$ which connects t_j and t_l.

4.2 Online adaptation

The final algorithmic step before a trajectory can be executed by the robot is to perform post-processing to ensure that it is time-continuous safe, i.e., that it lies entirely within $\mathcal{C}_{\text{free}}$. Although the library is constructed such that its trajectories avoid known obstacles, this step is crucial for robustness against obstacles that were unknown during library generation. We refer to this post-processing as *adaptation*.

There are a number of existing trajectory optimisation algorithms that are designed for similar purposes. In general, these algorithms require an initial seed trajectory as input, which they attempt to adapt to satisfy given constraints and maximise/minimise a given objective. In this work we use the TrajOpt (Schulman et al. 2014) algorithm with the modified library trajectory used as the seed trajectory.

Unfortunately, TrajOpt and similar trajectory optimisation approaches are not guaranteed to succeed in finding a solution and depend heavily on the choice of seed trajectory. The modified library trajectory already accounts for obstacles known at the time of library construction and is preferable to less informed choices, such as a straight line trajectory in configuration space. However, a fallback

Algorithm 2 Online Planner: Batch Query

```
1: Given an online environment and set of tasks, group tasks by subspace
2: Append home configuration to each group
3: Execute Lines 1–3 of Alg. 1 for each pair of tasks within each group
4: Construct weighted adjacency matrix for each group, using computed trajectory costs as weights
5: Compute optimal sequence for each group using TSP solver with associated weight matrix and start/end constrained to home configuration
6: Concatenate sequences
7: Execute Line 4 of Alg. 1 for each successive pair of tasks in the concatenated sequence
```

method remains necessary in case of failure due to obstacles discovered at execution time. The fallback method should be guaranteed to find a solution, since it is a method of last resort. HAP uses BIT* (Gammell et al. 2015), a well-known probabilistically complete planner, with the IK solution defined in the modified library trajectory used as its goal configuration input. Like all probabilistically complete methods, BIT* may fail to find a solution in a reasonable amount of time. HAP enforces a user-defined threshold and terminates execution if exceeded. This is the only condition in which HAP fails to produce an executable trajectory.

4.3 Additional operations for task sequencing in batch-query problems

The batch-query case is addressed by introducing several additional steps in Alg. 2. Given a set of online tasks T, we group tasks by subspace and describe how the single-query processes just described are used to find a minimal sequence of trajectories between them.

Each task $t \in T$ is assigned to one of the subspace mappings θ^i by querying the library. The tasks are partitioned into groups of equivalent subspace assignment, and the home configuration is appended to each group. The single-query process is then executed for all pairs of tasks within each group. The trajectory costs returned are stored as a weighted adjacency matrix. Thus each task group has an associated matrix where the weights correspond to the trajectory cost for each pair of tasks within the group, including the home configuration. These matrices are used as input to a TSP solver, which returns a task sequence for each group independently. The TSP solver is constrained such that each sequence begins and ends at the home configuration.

The sequences are concatenated in arbitrary order. This concatenation is always possible by construction of the sequences, which all start and end at the home configuration. Finally, the online adaptation process is executed for each pair of tasks in the concatenated sequence and the result is returned.

5 Trajectory library construction

In this section we detail how the maps $\{\theta^i\}_{i=1}^K$ and trajectory library are computed given a robotic manipulator, anticipated environment, and task space. Additionally, practical considerations are given that describe implementation specific details useful when applying HAP.

5.1 Algorithm overview

To construct the trajectory library, we are given an anticipated environment m and set of tasks $\hat{T}$ that are representative of online scenarios. An undirected graph G is created with nodes corresponding to tasks $t \in \hat{T}$ and edges formed via a maximum connection radius (see Sec. 5.3.7). Figure 3 shows an example G constructed over the m and $\hat{T}$ in Fig. 2(a). The trajectory library is then generated from G according to Alg. 3.

Algorithm 3 is initialised by assigning all nodes to $\hat{T}_{\text{open}}$. A node stays in $\hat{T}_{\text{open}}$ until a unique IK solution is assigned. While $\hat{T}_{\text{open}}$ is not empty, θ is found via a *generate map* algorithm procedure (Alg. 4). Algorithm 4 searches for a θ that minimizes the objective cost, the sum of all *minimum cost* paths $\pi^*(t_0, -)$ from a root node $t_0 \in G$ to all other nodes. That is,

$$J(\theta, t_0) = \sum_{t \in G \setminus t_0} g(\pi^*(t_0, t); \theta), \tag{4}$$

where the cost of any path $\pi(t_0, t_{N-1}) = \{\theta(t_0), \ldots, \theta(t_{N-1})\}$ of N nodes is defined as

$$g(\pi(t_0, t_{N-1}); \theta) = \sum_{n=0}^{N-2} d_C(\theta(t_n), \theta(t_{n+1})). \tag{5}$$

To ensure that trajectories have bounded lengths and do not involve large, unnecessary arm movements, θ is additionally constrained such that it is an approximate isometry, or an ϵ-Gromov-Hausdorff approximation (ϵ-GHA), defined below.

Definition 1. *The map $\theta : (\hat{T}, d_T) \to (\mathcal{C}, d_C)$ is an ϵ-Gromov-Hausdorff approximation if $\forall t_j, t_l \in \hat{T}$*

$$|d_T(t_j, t_l) - d_C(\theta(t_j), \theta(t_l))| < \epsilon \tag{6}$$

for some $\epsilon > 0$.

Depending on the value of ϵ, topology of m and robot kinematic structure, some nodes may still have undefined mappings after a single iteration of the algorithm. It may then be necessary to search for multiple ϵ-GHAs, $\{\theta^i\}_{i=1}^K$, that map a covering set of subspaces $\{\hat{T}^i\}_{i=1}^K \subset \hat{T}$ to a set of disjoint subspaces $\{\mathcal{C}^i\}_{i=1}^K \subset \mathcal{C}$. After the ϵ-GHA(s) is (are) found, we use the Dijkstra's algorithm to find minimum cost paths $\pi^*(t_j, t_l)$ pairwise between all tasks $t_j, t_l \in G$ using the fixed, unique IK solutions assigned by θ or $\{\theta^i\}_{i=1}^K$. The minimum cost paths are then stored in the trajectory library.

5.2 IK solution selection

The *generate map* algorithm as outlined in Alg. 4 is based on Dijkstra's algorithm and attempts to find a unique IK solution for each task in $\hat{T}$ such that the objective cost in (4) is minimised. It begins by assigning an undefined mapping to all nodes except for the root node which is mapped arbitrarily to one of its IK solutions. The rest of the procedure is carried out as in the original Dijkstra's algorithm with a priority

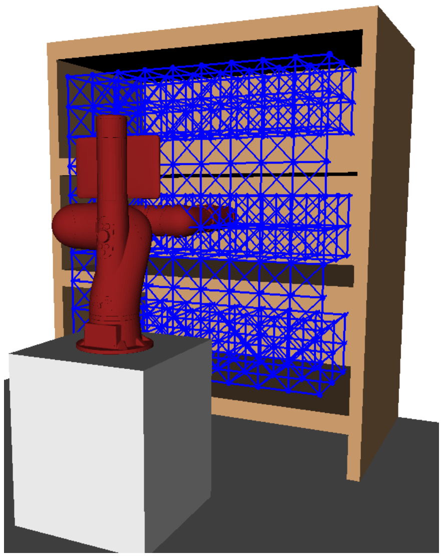


Figure 3. An undirected graph G constructed over discretised task space $\hat{T}$, where nodes are tasks in $\hat{T}$. Subspaces are searched for by traversing the graph and assigning an unique IK solution to each node such that all connected neighbours are close in configuration space.

queue $\mathbb{Q}$ (Barbehenn 1998), with modifications to the node expansion step where we compute the unique IK solution mapping. Here, the set of neighbouring nodes of t, $\mathcal{N}_t$, is run through the functions shown in Algs. 5 and 6.

In *get mapping*, if a node $u \in \mathcal{N}_t$ has already been assigned an IK solution q_u, this solution and the edge cost,

$$l(u,t) = d_C(\theta(u), \theta(t)), \tag{7}$$

are returned and the ϵ-GHA θ is not updated. Otherwise, the IK solution that gives the minimum $l(u,t)$ while satisfying the ϵ-GHA constraint in (6) is returned. The returned q_u and $l(u,t)$ are passed as a candidate to *update*. In *update*, if the candidate path cost from root node to $u \in \mathcal{N}_t$ is less than the current cost then q_u and the path cost are updated. Additionally, if u is not in $\mathbb{Q}$, it is added.

The above process is repeated for all IK solutions of the root node. If required, the algorithm is run K times to find multiple ϵ-GHAs. The resulting one or more ϵ-GHA(s) that minimise $J(\theta^i, t_0)$ are returned. All nodes are assigned an IK solution before being placed in the queue. It is also important to note that these assignments do not change during an iteration of the routine, as this could result in unstable behaviour and violation of the ϵ-GHA condition due to changing edge costs. However, path costs may change after finding shorter paths through G.

In contrast to the original Dijkstra's algorithm, all nodes are initialised with a non-infinite path cost c_{max}. This way, a candidate IK assignment is only accepted if it does not lead to high cost. Additionally, ϵ-GHAs with greater coverage will be favoured, particularly in earlier iterations.

5.3 Practical considerations

5.3.1 Encouraging exploration. While searching for ϵ-GHA(s), it is beneficial to bias the search toward the unexplored region of the task space. To encourage subspace exploration in subsequent iterations of the routine, a penalty

Algorithm 3 Search for ϵ-GHA maps and generate library

```
Input: environment $\boldsymbol{m}$ and poses $\hat{\boldsymbol{T}}$
Output: trajectory library $\hat{\boldsymbol{\pi}}$
function GenerateTrajectoryLibrary($m$, $\hat{T}$)
    $G \leftarrow$ gen_edges($\hat{T}$)
    $\hat{T}_{\text{open}} \leftarrow \hat{T}$
    $i \leftarrow 0$
    while $\hat{T}_{\text{open}} \neq \emptyset$ do
        $\hat{T}_{\text{root}} \leftarrow$ sample($\hat{T}_{\text{open}}$)
        $J(\theta^i, \hat{T}_{\text{root}}) \leftarrow \infty$
        for each $t_0$ in $\hat{T}_{\text{root}}$ do
            $J(\theta', t_0), \theta' \leftarrow$ GenerateMap($G, t_0$)
            if $J(\theta', t_0) < J(\theta^i, \hat{T}_{\text{root}})$ then
                $J(\theta^i, \hat{T}_{\text{root}}) \leftarrow J(\theta', t_0)$
                $\theta^i \leftarrow \theta'$
            end if
        end for
        for each $t$ in $G \mid \theta^i(t)$ not UNDEFINED do
            $\hat{T}_{\text{open}} \leftarrow \hat{T}_{\text{open}} \setminus \{t\}$
            $\hat{\pi}^i(t, -) \leftarrow$ Dijkstra's($G, t, \theta^i$)
        end for
        $i \leftarrow i + 1$
    end while
    return $\hat{\pi}$
end function
```

Algorithm 4 Search for candidate ϵ-GHA map

```
Input: graph $\boldsymbol{G}$, root node $\boldsymbol{t_0}$
Output: minimum sum of path costs $\boldsymbol{J(\theta', t_0)}$ and
    corresponding mapping $\boldsymbol{\theta'}$
function GenerateMap($G$, $t_0$)
    $J(\theta', t_0) \leftarrow \infty$
    for each $q$ in $Q(t_0)$ do
        for each $t$ in $G$ do
            $\theta(t) \leftarrow q$ if $t = t_0$, else UNDEFINED
            $g(\pi(t_0, t_0); \theta) \leftarrow 0$ if $t = t_0$, else $c_{\text{max}}$
        end for
        while $\mathbb{Q} \neq \emptyset$ do
            $t \leftarrow \text{argmin}_{t' \in \mathbb{Q}} g(\pi(t_0, t'); \theta)$
            $q_t \leftarrow \theta(t)$
            for each $u$ in $\mathcal{N}_t$ do
                if $\theta(u)$ UNDEFINED then
                    $q_u, l(u,t) \leftarrow$ GetMapping($u, q_t$)
                else
                    $q_u \leftarrow \theta(u)$
                    $l(u,t) \leftarrow d_C(q_t, \theta(u))$
                end if
                $g(\pi(t_0, u); \theta), \theta \leftarrow$ Update($\mathbb{Q}, q_u, u, t$)
            end for
        end while
        $J(\theta, t_0) = \sum_{t \in G \setminus t_0} g(\pi(t_0, t); \theta)$
        if $J(\theta, t_0) < J(\theta', t_0)$ then
            $J(\theta', t_0) \leftarrow J(\theta, t_0)$
            $\theta' \leftarrow \theta$
        end if
    end for
    return $J(\theta', t_0), \theta'$
end function
```

Algorithm 5 Get candidate mapping for neighbour node u

Input: Neighbouring node $\boldsymbol{u}$ and expanded node mapping $\boldsymbol{q_t}$

Output: candidate mapping $\boldsymbol{q_u}$ and resulting edge cost $\boldsymbol{l(u,t)}$

```
1:  function GETMAPPING($u$, $q_t$)
2:      $l(u,t) \leftarrow \infty$
3:      for each $p$ in $Q(u) \mid d_C(q_t,p) < \epsilon + d_T(u,t)$ do
4:          if $d_C(q_t,p) < l(u,t)$ then
5:              $q_u \leftarrow p$
6:              $l(u,t) \leftarrow d_C(q_t,q_u)$
7:          end if
8:      end for
9:      return $q_u$, $l(u,t)$
10: end function
```

Algorithm 6 Update neighbour path cost and node mapping

Input: Queue $\mathbb{Q}$, candidate mapping $\boldsymbol{q_u}$, neighbour node $\boldsymbol{u}$ and expanded node t

Output: updated path cost $\boldsymbol{g(\pi(t_0,u);\theta)}$ and updated IK assignment $\boldsymbol{\theta(u)}$

```
1: function UPDATE($\mathbb{Q}$, $q_u$, $l(u,t)$)
2:     if $l(u,t) + g(\pi(t_0,t);\theta) < g(\pi(t_0,u);\theta)$ then
3:         $\theta(u) \leftarrow q_u$
4:         $g(\pi(t_0,u);\theta) \leftarrow l(u,t) + g(\pi(t_0,t);\theta)$
5:         $\mathbb{Q} \leftarrow \mathbb{Q} \cup \{u\}$
6:     end if
7:     return $g(\pi(t_0,u);\theta)$, $\theta(u)$
8: end function
```

$\rho \cdot \omega(t)$ may be added to all edge costs passing through a node, where $\omega(t)$ is a count of how many times a node is assigned an IK solution, and ρ is a tunable parameter. As the penalties $\rho \cdot \omega(t)$ accumulate, previously mapped portions of the graph are not considered in later iterations of the algorithm as their path costs may exceed c_{max}. The algorithm then focuses on previously unmapped nodes to increase coverage of $\hat{T}$.

5.3.2 Adding a task-based cost. In the problem formulation we defined the motion planning and task sequencing objective in terms of trajectory length according to d_C, a metric on the configuration space. We thus assigned IK solutions to nodes in G via the *generate map* algorithm in Alg. 4, using edge costs as in (7). However, edge costs in G are not limited to this form. In fact, one can add a task-specific component $c(\theta(u),\theta(t))$ to (7) while maintaining bounded trajectory lengths. For example, a task-specific component may be introduced to penalise configurations near joint limits. Then, the ϵ-GHA found will ensure both a bounded trajectory length and that task-specific requirements are met. It is important to note that while the edge costs may be different to d_C the ϵ-GHA condition is still checked using the original d_C metric.

5.3.3 Ensuring smooth transitions between subspaces. As previously mentioned, when moving between subspaces the arm first moves back to a fixed home configuration. Based on a given home pose, the corresponding IK solution that minimises the distance to the average configuration q^0_{avg} of the first found subspace $\mathcal{C}^0$ is chosen. Explicity, q^0_{avg} is defined as

$$q^0_{\text{avg}} = \frac{1}{|\mathcal{C}^0|} \sum_{q \in \mathcal{C}^0} q. \tag{8}$$

Note that the home configuration can be computed online allowing for flexible home pose choice. To avoid large changes in configuration while returning to home, the subspaces are biased to be close to one other.

For $i \geq 1$ in Alg. 3 an additional penalty $\rho_s \cdot |\theta^i(u) - q^0_{\text{avg}}|$ with user-defined weighting ρ_s is added to the edge cost in (7). This way, chosen IK solutions are biased to be close in configuration space to q^0_{avg}. In addition, one can optionally enforce that the IK solution assignment for the root node t_0 is within some distance threshold, i.e., $|\theta^i(t_0) - q^0_{\text{avg}}| < \zeta$.

5.3.4 Balancing the number of subspaces. If many undefined node mappings remain after an iteration of Alg. 4, the trajectory library may not cover large regions of the task space. This may occur due to a failure to meet the ϵ-GHA condition, prioritisation of task-specific costs as in Sec. 7.5 or poor flexibility admitted by the robotic manipulator's natural kinematic configurations (demonstrated in Fig. B.1). This can have adverse impact on online planning if IK solutions of online tasks are far from trajectory library configurations, leading to failures or jerky trajectories. As such, Alg. 3 terminates only when $\hat{T}_{\text{open}} = \emptyset$ and a complete set of subspaces that fully cover the task space are found. Note that this condition may never be met. For example, a region requiring large configuration changes to switch to may have undefined mapping if a conservative ζ is chosen.

In practice a large number of subspaces is undesirable as it can slow trajectory library matching during online planning. Furthermore, frequent subspace switching can add cumbersome overhead to online execution. Thus, to balance a trade-off between coverage and planning/execution time the main loop in Alg. 3 can be terminated once a user-defined maximum number of subspaces have been found or until the ζ threshold cannot be satisfied. Alternatively, the loop can be terminated once a certain task-space coverage percentage has been achieved or when the size of the subspace found in an iteration falls below a set threshold.

5.3.5 Modifications for mobile bases. Greater coverage of the task space with fewer ϵ-GHAs can potentially be achieved by allowing the arm to be mobile. Having multiple ϵ-GHAs synergises well with a mobile manipulator. Instead of allocating an arbitrary number of ϵ-GHAs, one can choose a discrete number of base poses and assign ϵ-GHAs to each base pose.

When building the trajectory library for a mobile base, the *generate map* algorithm in Alg. 4 is run for all base poses in each iteration in Alg. 3. The pose that yields the lowest objective cost is allocated a ϵ-GHA. This base pose is then removed as a candidate in the subsequent iterations. This is repeated until all base poses have been assigned a subspace or until $\hat{T}_{\text{open}} = \emptyset$. All base poses are considered before allocating any mappings in order to find the base poses that give greater coverage early and lower objective costs. This way, base poses better suited for difficult to reach regions of the task space are reserved for later iterations.

For mobile base online execution, the subspace switching action consists of moving back to the home configuration before moving the base. Furthermore, when sequencing for the mobile base case we ignore the base movement cost when switching between subspaces. However, if this were important then sequencing the group execution order could be solved using another TSP with the base movement costs.

5.3.6 Environment design. An environment representation m can be given at trajectory library construction time to inform ϵ-GHA computation. Obstacles can be represented as a union of basic 3D shape primitives. For example, a bookshelf can be represented as a union of boxes. The shape and size is chosen such that a bounded volume is formed around static parts of the scene. In Fig. 2 an example of a bookshelf environment is modeled. In a real scenario the geometry of this model should remain roughly the same with only new objects appearing on shelves. To account for sensor noise, modelling inaccuracy and/or localisation error in the online scenario the bookshelf is inflated slightly. Inflation additionally encourages library trajectories to have greater clearance from the obstacles, increasing robustness to new unseen obstacles.

In this paper, m is defined manually for a range of obstacle configurations; however, a strategy could be developed to create these procedurally. Furthermore, although not in the scope of this paper, one could implement a learning algorithm to choose a trajectory library based on the obstacle configuration given in an online setting. Similar strategies have been used in previous work (Tallavajhula et al. 2016; Pan et al. 2014; Dey et al. 2013).

5.3.7 Task-space graph construction. An example of an undirected graph on $\hat{T}$ is visualised in Fig. 3, where nodes are tasks in $\hat{T}$ and edges are connections between tasks. The construction strategy of such a graph is flexible, however there are a few important considerations to highlight. Firstly, greater edge connection density allows for more diverse paths and greater node density increases the probability of time continuous safety being met by the trajectory library. Additionally, edges connect nodes in task space and hence a local connection strategy needs to be devised to ensure they are feasible in the configuration space. The connection strategy used in this paper is to connect nodes within a ball of specified radius in the workspace. To ensure feasibility, it is required that an IK solution in $\mathcal{C}_{\text{free}}$ exists for a discrete set of points along the edge connecting two nodes.

The $\hat{T}$ upon which the graph is built is a discretisation of the user-defined task space. In the context of this work it represents approximate poses that the arm is expected to plan to. An example discretisation strategy is visualised in Fig. 2(a). Poses here could represent abstract tasks such as pre-grasp points or camera viewpoints for active perception. The orientation of the poses in $\hat{T}$ should align roughly with the expected tasks. For example, in Fig. 2(a) all poses point forward into the bookshelf, a suitable construction for tasks such as grasping and scene reconstruction. Notice in the online scenario in Fig. 2(b) the task poses need not lie exactly on $\hat{T}$. However, performance may decrease the greater this discrepancy is. The poses in Fig. 2(a) are generated procedurally by defining a uniform graph of nodes across a volume bounding the bookshelf, however there exists many possible methods for generating these poses. For example, the manipulator could be teleoperated to various poses or the manipulator could be moved kinesthetically. That is, the human operator could move the end effector directly and store the poses. The requirement is the operator must generate poses such that no islands are formed in the graph. For example, in the bookshelf scenario in Fig. 2 there is a plane of poses in front of the bookshelf to ensure that there is connectivity between poses within the shelves. However, this could potentially be resolved in a post-processing step automatically, relieving the burden on the operator.

6 Analysis

In this section, we show that the trajectories found by Alg. 3 have bounded lengths d_C, and are hence efficient and free of jerky motion. This is because, as we show, ϵ-GHAs approximately preserve shortest paths between metric spaces, in our case the task and configuration spaces. We first verify that the map θ found by Alg. 3 is indeed an ϵ-GHA.

Theorem 1. θ is an ϵ-GHA. *θ found by Alg. 4 is an ϵ-GHA.*

Proof. Pick any $t \in \hat{T}$. Then $\forall u_1 \in \mathcal{N}_t$, by construction, we have

$$|d_C(\theta(t), \theta(u_1)) - d_T(t, u_1)| < \epsilon,$$

for some $\epsilon > 0$. Then, $\forall u_2 \in \mathcal{N}_{u_1}$, i.e. the next-nearest neighbours of t, we again have by construction

$$|d_C(\theta(u_1), \theta(u_2)) - d_T(u_1, u_2)| < \epsilon.$$

Similarly for all $(N-1)$th and Nth nearest neighbours of t,

$$|d_C(\theta(u_{N-1}), \theta(u_N)) - d_T(u_{N-1}, u_N)| < \epsilon.$$

Then, using the triangle inequality we get

$$\begin{aligned} &|d_C(\theta(t), \theta(u_N)) - d_T(t, u_N)| \\ &\leq |d_C(\theta(t), \theta(u_1)) - d_T(t, u_1)| + \\ &\textstyle\sum_{n=1}^{N-1} |d_C(\theta(u_n), \theta(u_{n+1})) - d_T(u_n, u_{n+1})| \\ &< \epsilon + (N-1)\epsilon \\ &= N\epsilon. \end{aligned}$$

As ϵ is arbitrary, taking ϵ to be $N\epsilon$ gives the required result. □

Furthermore, θ maps shortest paths in the workspace to paths in configuration space that are of bounded length. That is, *minimising geodesics* are approximately preserved under the mapping. Intuitively, minimising geodesics (referred to henceforth as geodesics for brevity) are a generalisation of “straight lines”, or shortest paths, in Euclidean space to more general spaces, defined below.

A metric space (X, d_X) is a set X equipped with a metric, or “distance function”, $d_X : X \times X \to \mathbb{R}$ that satisfies the axioms of positiveness, symmetry, and triangular inequality (Burago et al. 2001). Drawing upon concepts from metric geometry, we characterise geodesics on metric spaces as paths whose segment lengths sum to that of the whole path length. Formally,

Definition 2. Geodesics. *Given a metric space (X, d_X) with intrinsic metric d_X, a path $\gamma : [0,1] \to X$ is a geodesic iff:*

$$d_X(\gamma(0), \gamma(1)) = \sum_n d(\gamma(s_n), \gamma(s_{n+1})), \tag{9}$$

for any $\{s_n\} \subset [0,1]$.

We now show that an ϵ-GHA, and thus θ found by HAP, preserves geodesics approximately.

Theorem 2. ϵ-GHAs preserve geodesics. *Let (X, d_X) and (Y, d_Y) be two metric spaces, and $\theta : X \to Y$ an ϵ-GHA. Then, for any geodesic γ on X, we have*

$$|d_Y(\theta(\gamma(0)), \theta(\gamma(1))) - \sum_n d_Y(\theta(\gamma(s_n)), \theta(\gamma(s_{n+1})))| \\ \le (N+1)\epsilon. \tag{10}$$

In other words, the path $\theta(\gamma(s)) : [0,1] \to Y$ is $(N+1)\epsilon$ away from being a geodesic in the configuration space.

Proof. Applying the ϵ-GHA condition to the end-points of γ, we have

$$|d_Y(g(\gamma(0)), g(\gamma(1))) - d_X(\gamma(0), \gamma(1))| \le \epsilon. \tag{11}$$

Because γ is a geodesic, we can replace the $d_X(\gamma(0), \gamma(1))$ term in (11) with the sum of lengths along any test points $\{s_1, ..., s_N\}$,

$$|d_Y(g(\gamma(0)), g(\gamma(1))) - \sum_n d_X(\gamma(s_n), \gamma(s_{n+1}))| \le \epsilon. \tag{12}$$

Using the ϵ-GHA condition on summands individually, we have

$$\begin{aligned} &\sum_n d_Y(g(\gamma(s_n)), g(\gamma(s_{n+1}))) - (N+1)\epsilon \\ &\le d_Y(g(\gamma(0)), g(\gamma(1))) \\ &\le \sum_n d_Y(g(\gamma(s_n)), g(\gamma(s_{n+1}))) + (N+1)\epsilon. \end{aligned} \tag{13}$$

$N+1$ arises because there are N epsilons inside the sum and one outside. □

7 Experiments

This section presents several sets of experimental results that demonstrate and evaluate HAP's performance in simulation using high-quality kinematic models of robot manipulators. For comparison, we also present results obtained using alternative methods

We consider both single-query and batch-query problems with a typical fixed-base 7-DOF manipulator model. Further, we extend the evaluation in two ways. The first extension is to show that the HAP framework is not limited to fixed manipulators and can also be used for mobile manipulators, i.e., a manipulator mounted to a mobile base. The second extension is to show the robustness of HAP trajectories to small perturbations in goal pose by adding a task-space control step, where the end effector must achieve a given approach pose and then perform visual-servoing to arrive at the given goal pose.

We begin by describing the experimental setup, including environment models, robot models, and comparison methods. Implementation details are also provided. We then present single-query and batch-query experiments for fixed and mobile bases, followed by task-space control/visual servoing experiments for a mobile base.

Variant	Sequencer (batch query)		Planner		
	HAP	Naive	TrajOpt Seed		BIT*
			HAP	Naive	
HAP-Full	✓		✓		✓
Hybrid-Full		✓	✓		✓
Naive-Full		✓		✓	✓
HAP-Opt	✓		✓		
Naive-Opt		✓		✓	

Table 1. Planner variants used for experiments with the fixed-based manipulator model.

Variant	Base Disabled	ϵ-GHAs		
		Single	Multiple	N/A
HAP-Mobile			✓	
HAP-Multi	✓		✓	
HAP-Single	✓	✓		
Naive-Mobile				✓
Naive-Static	✓			✓

Table 2. Planner variants used for experiments with the mobile-base manipulator model.

Variant	Planning time (s)	Execution time (s)	Success rate (%)
	Fixed base		
HAP-Full	$\mathbf{0.46 \pm 0.62}$	1.62 ± 0.79	**94.00**
Naive-Full	0.92 ± 0.97	$\mathbf{1.49 \pm 1.22}$	79.60
	Mobile base		
HAP-Multi	$\mathbf{0.44 \pm 0.42}$	1.58 ± 1.02	**89.40**
Naive-Static	1.13 ± 0.98	$\mathbf{1.40 \pm 1.45}$	72.60
	Mobile base (subset)		
HAP-Multi	$\mathbf{0.20 \pm 0.14}$	$\mathbf{0.61 \pm 0.28}$	**100.00**
Naive-Static	0.91 ± 0.97	1.16 ± 1.59	87.25

Table 3. Results for single-query experiments with the fixed-base model, the mobile-base model, and a subset of mobile-base trials where start and goal poses occupy the same subspace. Times are reported as the average and standard deviation computed over the set of trials. HAP consistently achieved superior planning time and success rate.

7.1 Experimental setup

Experiments consist of sets of randomised trials where a planning method is repeatedly executed with a given environment and robot model. Tasks are sampled uniformly at random from the environment's task space. To discourage dense sampling, we discard task samples that lie within a threshold of 0.15 m from a previous sample.

We perform 500 trials for each planning method in single-query problems, and 50 trials in batch-query problems with a batch size of 10. The following metrics are evaluated:

- *Planning success rate* refers to the percentage of tasks for which the motion planner succeeded in finding a trajectory in $\mathcal{C}_{\text{free}}$.

- *Planning time* refers to the computation time taken for the motion planners to compute a plan for a task.
- *Execution time* refers to the time taken for a successful trajectory execution, assuming the arms are operating at their maximum joint velocities.
- *Maximum trajectory jerk* refers to the maximum joint jerk norm of an executed trajectory. This is computed numerically using the finite difference method.
- *Sequencing time (batch-query only)* refers to the time taken to match library trajectories, modify them and then compute a sequence, including the IK solution computation for each task for both the naive and HAP planning methods.

7.1.1 Robot models. The 7-DOF fixed-base manipulator is a model of the *Sawyer* robot originally produced by Rethink Robotics. The mobile-base manipulator is a model of Universal Robots' UR5 collaborative robot arm fixed to a base that can translate in the plane.

7.1.2 Comparison methods: fixed base. The set of comparison methods for experiments with the fixed-based model consists of the following five variants: HAP-Full, Hybrid-Full, Naive-Full, HAP-Opt, and Naive-Opt. Their properties are summarised in Table 1.

There are two sequencing method options for batch-query problems, labelled *HAP* and *Naive*. HAP refers to the method proposed in this work. The Naive method sequences tasks according to task-space distance only and ignores configuration space.

There are also two options for planning methods, labelled *TrajOpt Seed* and *BIT**. These options refer to the online adaptation step in both single-query and batch-query problems. The TrajOpt (Schulman and the Robot Learning Lab 2013) algorithm is executed initially in all variants, as proposed in the HAP framework, although the seed trajectory provided to TrajOpt is varied. The HAP seed suboption is the retrieved library trajectory; the Naive seed suboption is the straight-line trajectory between start and goal configurations. If TrajOpt fails to find a plan, variants with the BIT* (Kavraki Lab 2017) option (suffixed with -Full) subsequently execute the BIT* algorithm, as proposed, with a time limit of two seconds.

7.1.3 Comparison methods: Mobile base. Comparison methods for experiments with the mobile-base model are: HAP-Mobile, HAP-Multi, HAP-Single, Naive-Mobile, Naive-Static. HAP-Mobile, HAP-Multi, and HAP-Single use the same options as HAP-Full in Table 1. Likewise, Naive-Mobile and Naive-Static use the same options as Naive-Full.

The variants differ in that the mobile base can be enabled (allowed to move) or disabled (static). They also differ in whether single or multiple ϵ-GHAs are generated during trajectory library construction. These properties are summarised in Table 2. The computed subspaces for HAP-Multi and HAP-Mobile are shown in Figs. B.2 and B.3.

The Naive-Mobile variant further differs in how tasks are allocated to base positions for experimental trials. Without HAP's automatic subspace generation and allocation for mobile bases (see Sec. 5.3.5), tasks are manually organised into three different divisions as shown in Fig. D.1 and D.2.

7.1.4 Environments and trajectory library construction. All single-query and batch-query experiments use the bookshelf environment shown in Fig. 2(b). For all experiments with the fixed-base model, a single ϵ-GHA is defined in the library. The home configuration is the home pose IK solution closest to the average configuration of this ϵ-GHA. Experiments with the mobile-base use single or multiple ϵ-GHAs as defined by the comparison method. The home configuration is defined similarly except with the average configuration of the first found ϵ-GHA.

7.2 Implementation details

In implementing Alg. 3, the number of root nodes $\hat{T}_{\text{root}} = 10$. For all multiple ϵ-GHA experiments the algorithm is terminated once $i = 5$. In implementing Algs. 4-6, all nodes are initialised with path cost $c_{\max} = 5.0$. The subspace exploration penalty $\rho = 2.0$. The subspace distance biasing penalty $\rho_s = 0.02$. Experiments with the mobile-base model use $\epsilon = 0.35$, and those with the fixed-base model use $\epsilon = 0.85$. Distance d_C is defined as L_∞, and d_T as L_2.

For the online planner (Alg. 1), the number of closest neighbours used when matching library tasks is $k = 10$. The thresholds used for terminating the search over mappings when matching a library configuration are L_2 distance 0.7 for the bookshelf environment and 0.5 radians for task-space control experiments. Google's or-tools (Google 2017) package is used as the TSP-solver in (3).

The home pose used is shown in Fig. 2(b). In mobile-base experiments, a discrete set of possible base positions is defined uniformly along the y-axis, parallel to the bookshelf.

7.3 Single-query results

Results for single-query experiments are provided in Table 3. This set of experiments compares HAP's performance to one of the naive variants. To more closely examine performance with the mobile-base model, we report results for a subset of trials in addition to the full set. This subset corresponds to trials where the start and goal poses lie in the same subspace, a condition which occurred in approximately 30% of trials in the full set.

HAP outperformed the naive baseline in planning time and success rate. Although HAP's average execution time is slightly worse, the lower standard deviation indicates better consistency across tasks.

The percentage of tasks in which HAP failed to produce a trajectory is explained by the presence of obstacles introduced after library construction or its sparseness. This behaviour is expected because we intentionally chose parameters that would reveal the algorithm's limits. In practice, success rate can be improved by increasing the size of the trajectory library or by increasing the maximum time allocated to the fallback planner (here, BIT*).

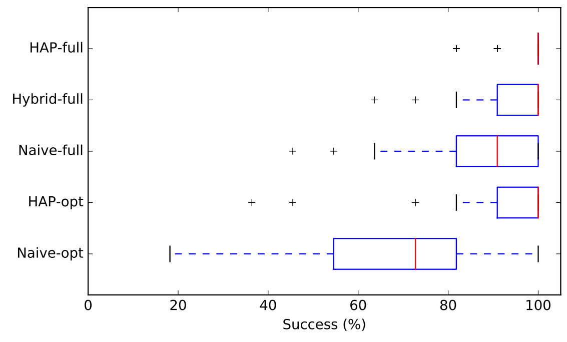
Further, our expectation of the benefit of identifying subspaces is confirmed. When isolating results to the subset of trials that lie within a single subspace we see that HAP's success rate is 100% with a large (over $2\times$) speedup in planning and execution times.

7.4 Batch-query results

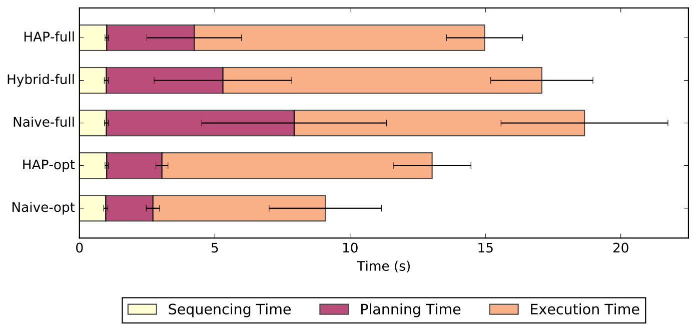
7.4.1 Fixed base. Results for batch-query experiments with the fixed-based model are provided in Fig. 4 for all algorithm variants. The box-and-whisker plots in Fig. 4a show that HAP achieved a favourable success rate relative to baseline methods, as expected. Planning and execution times shown in Fig. 4b are for the entire batch of 10 tasks. Although these results appear to indicate superior performance for HAP-opt and Naive-opt, these data describe successful trials only and should be interpreted in conjunction with corresponding planning success rates. Sequencing times are consistent, as expected, since the same TSP-solver is used by all variants.

Performance differences among variants are more distinct in the trajectory quality comparison provided in Table 4, which reports maximum jerk. HAP variants (HAP-Full and HAP-Opt) achieved roughly 50% lower average maximum jerk values than the naive variants.

7.4.2 Mobile base. Corresponding results for the mobile-base model are provided in Fig. 5. In these experiments, trajectories involve motion of the manipulator's base in addition to its joints. Performance is consistent with previous

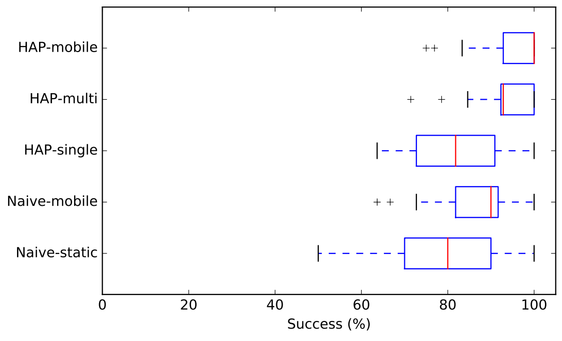


(a) Planning success rates

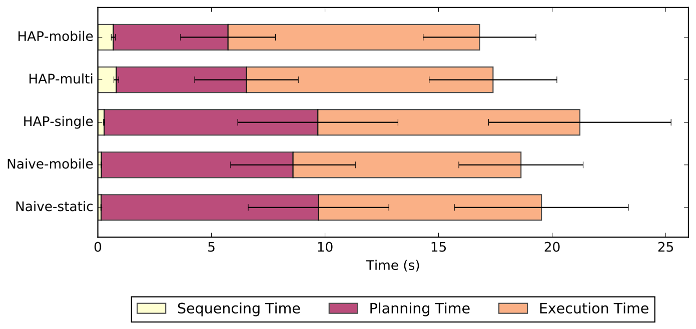


(b) Time performance metrics

Figure 4. Planning success rate (a) and time (b) for batch-query experiments with the fixed-base model. Mean and standard deviation for sequencing, planning, and execution time refers to a batch size of 10. HAP variants achieved higher planning success rate and faster plan times compared to the baseline naive variants while achieving similar or better execution times.



(a) Planning success rates



(b) Time performance metrics

Figure 5. Planning success rate (a) and time (b) for batch-query experiments with the mobile-base model. Results are consistent with fixed-base experiments shown in Fig. 4.

HAP-Full	189.84 ± 70.84
Hybrid-Full	$\mathbf{182.75 \pm 74.61}$
Naive-Full	412.53 ± 283.26
HAP-Opt	191.39 ± 71.15
Naive-Opt	368.46 ± 256.03

Table 4. Maximum trajectory jerk values (rad · s^{-3}) for batch-query experiments with the fixed-base model. Mean and standard deviation are computed over all successful trajectories. HAP variants consistently produced smoother (lower maximum jerk) trajectories.

HAP-Mobile	500.91 ± 167.08
HAP-Multi	610.33 ± 168.80
HAP-Single	$\mathbf{380.30 \pm 211.30}$
Naive-Mobile	524.19 ± 508.40
Naive-Static	787.13 ± 699.05

Table 5. Maximum trajectory jerk values (rad · s^{-3}) for batch-query experiments with the mobile-base model. Results are consistent with the fixed-base model reported in Table. 4.

Environment Instance	1 (original)	2	3	4	5	6
	Batch-query, fixed base, HAP-Full					
Success rate (%)	97.09 ± 5.59	97.09 ± 5.87	97.09 ± 5.59	95.27 ± 6.36	94.55 ± 9.09	95.64 ± 6.36
Plan times (s)	0.29 ± 0.16	0.30 ± 0.15	0.33 ± 0.16	0.37 ± 0.20	0.42 ± 0.21	0.38 ± 0.20
Execution times (s)	1.01 ± 0.15	0.98 ± 0.15	1.02 ± 0.20	1.02 ± 0.17	1.05 ± 0.19	1.07 ± 0.23
Sequence times (s)	1.01 ± 0.06	1.01 ± 0.06	1.01 ± 0.08	1.01 ± 0.09	1.06 ± 0.08	1.02 ± 0.08
	Batch-query, mobile base, HAP-Mobile					
Success rate (%)	95.20 ± 6.68	94.57 ± 5.55	93.46 ± 6.56	93.86 ± 5.98	93.04 ± 5.40	94.76 ± 4.96
Plan times (s)	0.37 ± 0.16	0.42 ± 0.18	0.43 ± 0.19	0.43 ± 0.18	0.43 ± 0.19	0.37 ± 0.13
Execution times (s)	0.84 ± 0.15	0.87 ± 0.19	0.85 ± 0.19	0.88 ± 0.15	0.86 ± 0.20	0.81 ± 0.13
Sequence times (s)	0.68 ± 0.09	0.70 ± 0.09	0.70 ± 0.10	0.72 ± 0.08	0.71 ± 0.09	0.69 ± 0.08

Table 6. HAP validation results in six instances of the bookshelf environments where object locations are randomly perturbed. Performance is consistent across all environment instances.

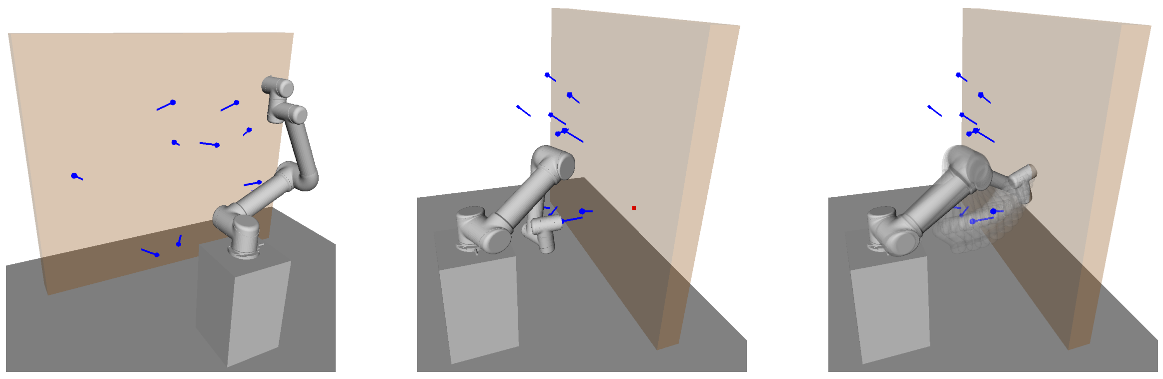

(a) Task-space control experiment scenario **(b)** Mobile-base model at approach pose **(c)** Mobile-base model at target pose after servoing

Figure 6. Task-space control experiments: (a) shows the experimental setup and an instance of tasks, (b) shows the arm at the approach pose (blue vector) facing the initial estimate of target position (red dot), (b) shows the trajectory produced by the PBVS controller.

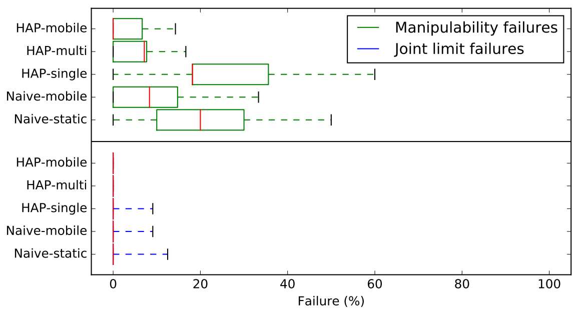


Figure 7. Control failure rates for task-space control experiments. HAP-Mobile and HAP-Multi achieved higher success rates than baseline methods.

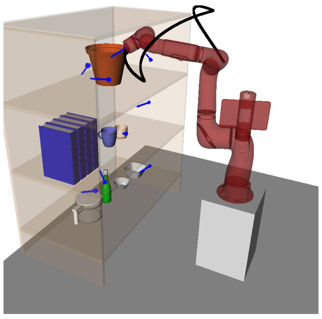
experiments, despite this increase in complexity. Notably, total time is increased in variants where base motion is disabled (HAP-Multi, HAP-Single, Naive-Static). This trend indicates that HAP is able to exploit base motion to generate shorter overall trajectories. Sequencing time is increased in variants that use multiple subspace maps, as expected.

Path quality statistics shown in Table 5 are also consistent with previous experiments. The availability of base motion is evident in HAP-Mobile and Naive-Mobile, which achieved lower maximum jerk values.

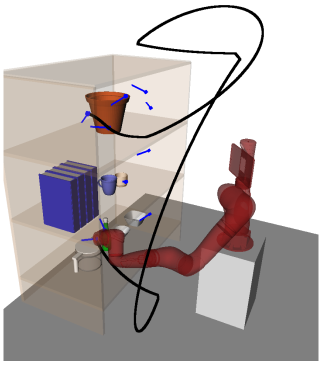
Effects related to the use of multiple subspace mappings are also examined in this set of experiments. HAP-Mobile and HAP-Multi use libraries constructed to allow multiple subspaces, whereas HAP-Single is limited to one subspace.

Concatenated trajectories pass through the home position when moving between subspaces. HAP-Mobile generated this behaviour 2.74 times per batch on average. The average is 2.88 for HAP-Multi, and 1.66 for Naive-Mobile. HAP-Single uses a single subspace and thus produced no subspace transitions. Planning success rate is generally inversely proportional, confirming our expectation that multiple subspaces are beneficial. Further, this pattern supports our expectation that automatically generated subspaces are more effective than the manually generated divisions used by Naive-Mobile.

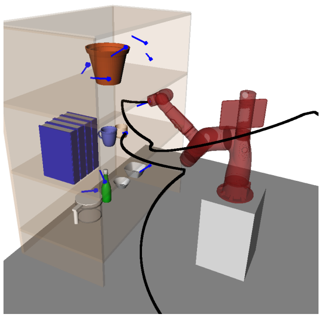
7.4.3 Varied obstacle configurations. Having compared HAP's performance to alternative methods within a given environment, we now report results that evaluate performance across a number of different environments. We



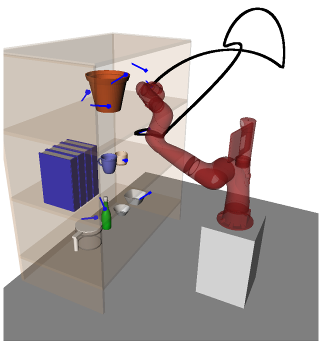
(a) $\sigma_{Naive}[0:1]$



(b) $\sigma_{Naive}[3:4]$

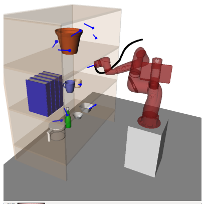


(c) $\sigma_{Naive}[6:8]$

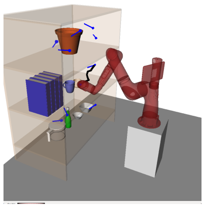


(d) $\sigma_{Naive}[8:9]$

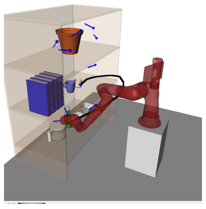
Figure 8. Example subset of a Naive-Full trajectory sequence for Sawyer bookshelf experiment. The trajectories for the tasks shown exhibit long and jerky trajectories. (c) skips a task in the sequence since it failed to plan to task 7 (bottom right shelf row).



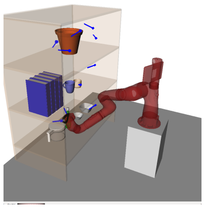
(a) $\sigma_{HAP}[0:1]$



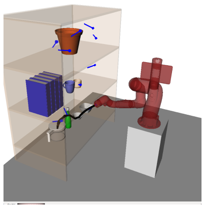
(b) $\sigma_{HAP}[1:2]$



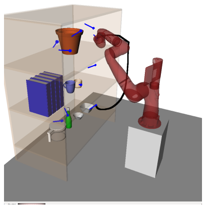
(c) $\sigma_{HAP}[2:3]$



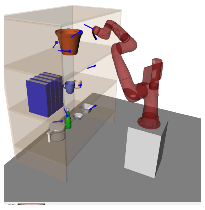
(d) $\sigma_{HAP}[3:4]$



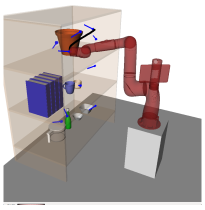
(e) $\sigma_{HAP}[4:5]$



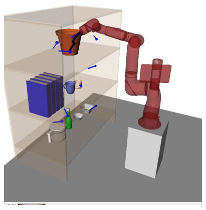
(f) $\sigma_{HAP}[5:6]$



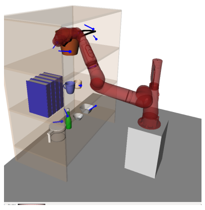
(g) $\sigma_{HAP}[6:7]$



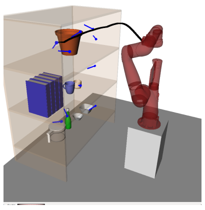
(h) $\sigma_{HAP}[7:8]$



(i) $\sigma_{HAP}[8:9]$



(j) $\sigma_{HAP}[9:10]$



(k) $\sigma_{HAP}[10:0]$

Figure 9. Example HAP-Full trajectory sequence for fixed-base model bookshelf experiment. All tasks are successfully planned and trajectories appear to be consistently short and smooth.

randomly perturbed the arrangement of objects (obstacles) in the bookshelf environment used previously to generate five additional instances of this environment.

Table 6 reports results for the batch-query case using the fixed-based model and HAP-Full, and for the batch-query case using the mobile-base model and HAP-Mobile. The original bookshelf environment is labelled as Environment Instance 1, and the other instances are arbitrarily enumerated. Performance is consistent across these environments as expected.

7.5 Task-space control

The third set of experiments extends our evaluation of HAP towards a typical application scenario where the planned trajectory positions the end effector in order to perform Position-Based Visual Servoing (PBVS) in task space. The goal pose input to HAP is an approach pose that lies at a fixed offset from the target. After the resulting trajectory is executed, a task-space control procedure is used to move the end effector in a linear motion towards the target. These experiments illustrate how task-based cost can be used to position the end effector in a way that facilitates subsequent control behaviour.

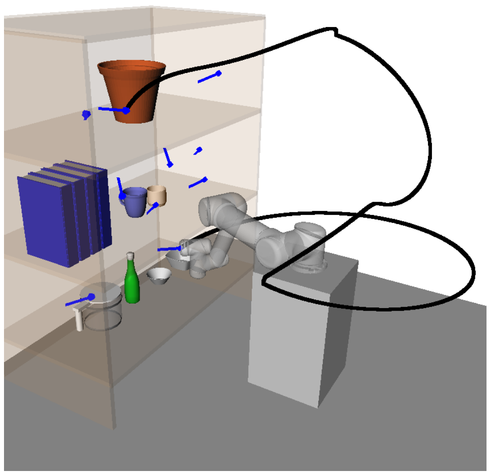
The experimental environment is shown in Fig. 6. The $\hat{T}$ used for the trajectory library generation and subspace allocations for the experiments are shown in Appendices C.1 and C.2. $\hat{T}$ is constructed by computing approach poses from expected target positions.

PBVS is implemented by computing desired linear and angular task-space velocities and then mapping these to joint velocities via the pseudo-inverse of the Jacobian. Task-based cost (Sec. 5.3.2) is defined as the weight sum

$$c(\theta(u),\theta(t)) = \xi_{\text{lim}}\frac{1}{P}\sum_{n=1}^{P}\tanh\left(2\left|\frac{\theta(u)_n - q_n^{\text{lower}}}{q_n^{\text{upper}} - q_n^{\text{lower}}} - \frac{1}{2}\right|\right) + \xi_{\text{manip}}\kappa(\mathbf{J}), \tag{14}$$

where $\kappa(\mathbf{J})$ is the condition number of the manipulator's Jacobian, $\mathbf{J}$. The first term penalises configurations near the joint limits where q_n^{lower} and q_n^{upper} are the lower and upper limits of the n-th manipulator joint. The second term encourages configurations with higher manipulability. We penalise the condition number $\kappa(\mathbf{J})$ because singularity occurs when $\kappa(\mathbf{J})$ approaches ∞ (Klein and Blaho 1987). Weight ξ_{lim} is set to 0.5 and ξ_{manip} to 0.1.

To perform PBVS in simulation, we add Gaussian noise to the target's position and orientation that decreases linearly


(a) $\sigma_{Naive}[1:3]$

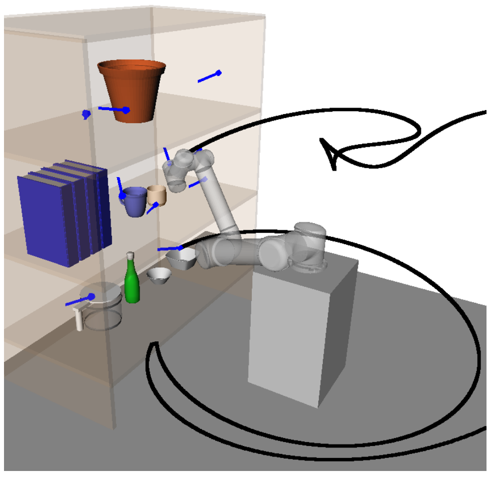
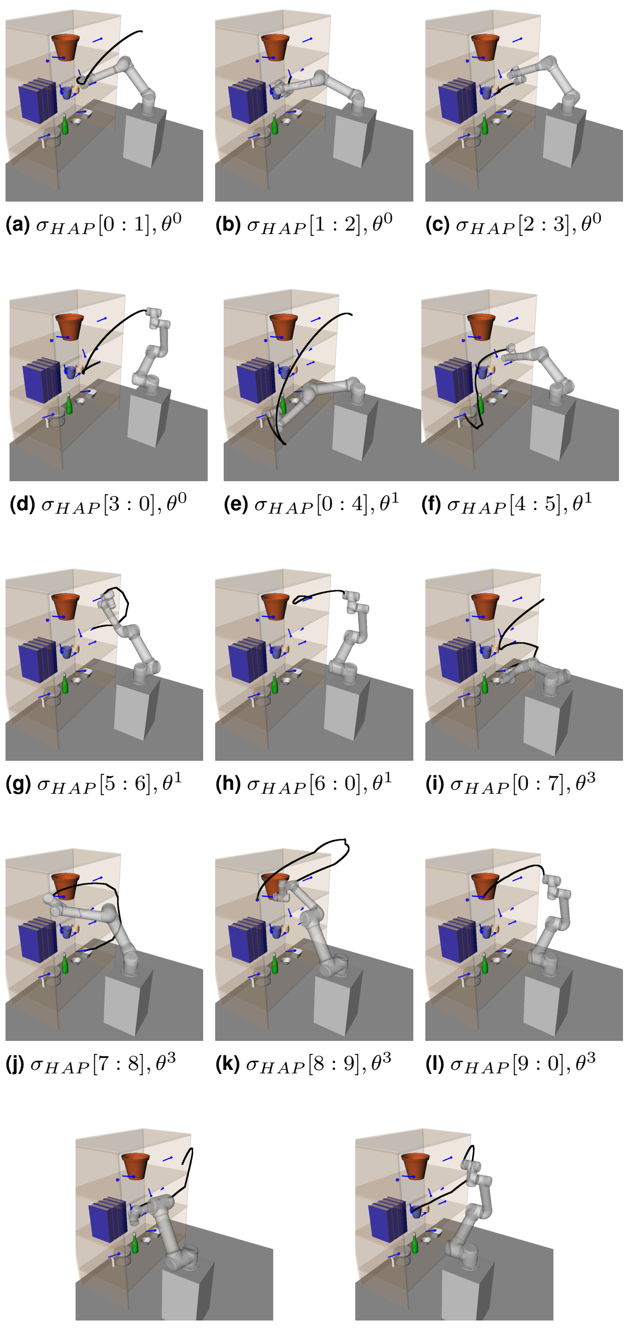

(b) $\sigma_{Naive}[3:5]$

Figure 10. Example subset of a Naive-Mobile trajectory sequence for mobile-base model bookshelf experiment. The trajectories for the tasks shown exhibit long jerky trajectories. Both (a) and (b) skip a task in the sequence since they failed to plan to two tasks in the middle and lower shelf rows.


(a) $\sigma_{HAP}[0:1], \theta^0$ (b) $\sigma_{HAP}[1:2], \theta^0$ (c) $\sigma_{HAP}[2:3], \theta^0$

(d) $\sigma_{HAP}[3:0], \theta^0$ (e) $\sigma_{HAP}[0:4], \theta^1$ (f) $\sigma_{HAP}[4:5], \theta^1$

(g) $\sigma_{HAP}[5:6], \theta^1$ (h) $\sigma_{HAP}[6:0], \theta^1$ (i) $\sigma_{HAP}[0:7], \theta^3$

(j) $\sigma_{HAP}[7:8], \theta^3$ (k) $\sigma_{HAP}[8:9], \theta^3$ (l) $\sigma_{HAP}[9:0], \theta^3$

(m) $\sigma_{HAP}[0:10], \theta^4$ (n) $\sigma_{HAP}[10:0], \theta^4$

Figure 11. Example HAP-Mobile trajectory sequence for mobile-base model bookshelf experiment. The assigned ϵ-GHA indexes for each task pair are shown to highlight when subspace switching occurs. All tasks are successfully planned and trajectories appear to be consistently short and smooth.

as the end effector approaches the target. Tasks are generated such that both the target pose and approach pose have at least one valid IK solution.

We performed 50 trials with a batch size of 10 using the same set of comparison methods used in previous batch-query experiments for the mobile-base model. Two failure conditions are of interest. The first is referred to as joint limit failure and occurs when any joint reaches a threshold of 0.01 radians from its limit. The second is manipulability failure and is defined by the condition $1/\kappa(\mathbf{J}) < 0.015$, indicating that the arm is approaching a singularity.

Results are plotted in Fig. 7. HAP-Multi and HAP-Mobile achieved low failure rates. Naive-Mobile, Naive-Static, and HAP-Single produced high rates of task-space control failures in particular. This pattern is consistent with the other results for the mobile-base model where multiple subspaces contributed to greater planning success.

8 Discussion

This section discusses properties of trajectories generated by HAP in relation to baseline methods. We provide examples that illustrate the benefits of our choice of seed trajectories for online adaptation and of our method for generating subspace mappings in batch-query problems.

Naive heuristics for planning, such as Euclidean distance in workspace, do not account for the nonlinear motion of high-DOF robotic manipulators. The ϵ-GHAs computed by HAP account for this nonlinearity by considering prior

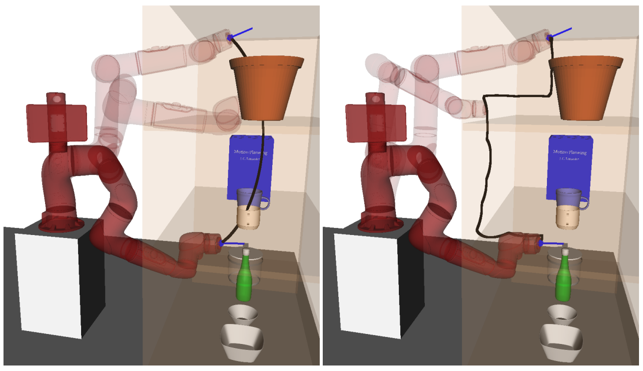

(a) Naive trajectory seed **(b)** HAP trajectory seed

Figure 12. Side view of bookshelf environment showing comparison of (a) naive planner's initial trajectory seed and (b) HAP's. Since HAP has prior knowledge of the environment it seeds with a trajectory that moves away from the shelves before moving into another row.

knowledge such as the kinematic structure and environment obstacles. As a result, HAP achieves improved performance, exemplified in Fig. 9 (Extension 1) where HAP begins tasks in the middle shelf and sequences tasks according to shelf rows. In contrast, the naive planner executes an inefficient sequence depicted in Fig. 8 (Extension 1), beginning in the top shelf, continuing tasks in other shelves and returning to the top shelf to complete the sequence.

Furthermore, naive trajectories can exhibit excessive jerk as the arm approaches awkward goal configurations along the sequence. In comparison, HAP maintains smooth trajectories over the entire sequence by biasing trajectories to the mapped subspaces. In Fig. 8(c), the naive planner failed in one of the tasks in the lower shelf.

The mobile base scenario shown in Figs. 10 and 11 (Extension 2) further demonstrates the benefit of our proposed approach. HAP allocates tasks on the lower shelf to a base pose that allows for more space for the arm to move (Fig. 11(j)) while still efficiently grouping tasks in the other rows. Notably, the naive method fails to plan to one such task on the lower shelf.

Interestingly, the trajectory in Fig. 11(k) required a large rotation about the wrist joint. This is due to the pot object in the top shelf row, which was not known during trajectory library construction. This example indicates that splitting or merging of subspaces may be a direction of future work that could improve the robustness of the framework.

Another key advantage of our approach over naive methods is the utilisation of trajectory initialisation that is informed by an *a priori* model of the environment. As illustrated in Fig. 12(a), the naive straight line trajectory initialisation passes through both the bookshelf and objects on the shelves. This is a poor initialisation for trajectory optimisation as can lead to local minima where links are in collision with multiple objects, or where multiple links have inconsistent gradient directions (Schulman et al. 2014). As a consequence, this example failed during TrajOpt execution.

The initialisation computed by HAP, shown in Fig. 12(b), allows the arm to move away from the shelves. This initialisation facilitates execution of TrajOpt, even though it may initially be in collision with other unseen objects on the bookshelf.

9 Conclusion and future work

We have presented a new motion planning framework for robotic manipulators that is designed to perform efficiently in practice given a user-defined task space. The framework automatically constructs a library of trajectories that move the manipulator's end effector from one pose to another within this task space, and uses this library to quickly produce motion plans online. The library is constructed by finding a set of subspaces with associated ϵ-Gromov-Hausdorff approximations that guarantee short trajectories of bounded length which also can be concatenated smoothly. A distinctive feature of our method is its usefulness in batch-query scenarios where an unordered set of goal poses is provided and the algorithm is free to choose a favourable sequence of minimum total cost. Because all precomputation is performed automatically, our work helps to enable rapid deployment of manipulators for various applications based on the task-space definition.

We have evaluated our framework in several experimental scenarios with 6-DOF and 7-DOF manipulators on fixed and mobile bases, including point-to-point motions and long sequences in a bookshelf environment. Results showed notable performance improvement over baseline methods in planning time, planning success rate, and smoothness measured by jerk. Experiments with manipulators mounted to a mobile base demonstrate the generality of our method for a variety of hardware configurations.

Our results motivate several important avenues of future work. It would be interesting to explore methods that would adapt the subspaces online in response to changes in the environment, potentially using online domain adaptation techniques developed for modelling laser scans (Tompkins et al. 2020) to handle changes in environment topology. A related idea would be to learn a distribution of library trajectories conditioned on the environment, akin to work that predicts pedestrian trajectories by learning a conditional multi-modal distribution (Zhi et al. 2021). The ϵ-GHAs could be used to inform choice of alternative local planners or controllers such as RMPflow (Cheng et al. 2021), which would benefit from knowledge of which regions of the task space are approximate isomorphisms to configuration space.

Finally, we are specifically interested in applying our work to cobot systems by investigating alternative metrics for trajectory cost. For example, a range of metrics have been proposed that aim to generate natural, human-like motion (Liarokapis et al. 2017; Gulletta et al. 2020; Gäbert et al. 2021). Other work aims to mimic human posture (Jaquier et al. 2021) by learning a manipulability matrix representation constrained to the space of symmetric positive definite (SPD) matrices. In the context of a manipulator, SPD matrices can be parameterised via a convex cone manifold embedded in $\mathbb{R}^6$. HAP could be used in a similar way to find ϵ-GHA mappings that potentially capture more complex geometric structures than SPD matrices which could then be used to bias trajectories towards more natural human behaviour.

Funding

This research is supported by Australian Government Research Training Program (RTP) Scholarships and the University of Technology Sydney.

References

Alatartsev S, Mersheeva V, Augustine M and Ortmeier F (2013) On optimizing a sequence of robotic tasks. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 217–223.

Alatartsev S, Stellmacher S and Ortmeier F (2015) Robotic task sequencing problem: A survey. *Journal of intelligent & robotic systems* 80(2): 279–298.

Arslan O and Tsiotras P (2013) Use of relaxation methods in sampling-based algorithms for optimal motion planning. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 2421–2428.

Bac CW, Hemming J, Tuijl B, Barth R, Wais E and Henten EJ (2017) Performance evaluation of a harvesting robot for sweet pepper. *Journal of Field Robotics* 34(6): 1123–1139.

Barbehenn M (1998) A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. *IEEE Transactions on Computers* 47(2): 263.

Burago D, Burago ID, Burago Y, Ivanov S, Ivanov SV and Ivanov SA (2001) *A course in metric geometry*, volume 33. American Mathematical Society.

Cheng CA, Mukadam M, Issac J, Birchfield S, Fox D, Boots B and Ratliff N (2021) Rmpflow: A geometric framework for generation of multitask motion policies. *IEEE Transactions on Automation Science and Engineering* .

Choset HM (2005) *Principles of robot motion: theory, algorithms, and implementation*. MIT press.

Dey D, Liu TY, Hebert M and Bagnell JA (2013) Contextual sequence prediction with application to control library optimization. In: *Proceedings of Robotics: Science and Systems (RSS)*.

Dogar MR, Hsiao† K, Ciocarlie† M and Srinivasa SS (2013) Physics-based grasp planning through clutter. In: *Proceedings of Robotics: Science and Systems (RSS)*.

Driess D, Ha JS and Toussaint M (2020) Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. In: *Proceedings of Robotics: Science and Systems (RSS)*.

Elbanhawi M and Simic M (2014) Sampling-based robot motion planning: A review. *IEEE Access* 2: 56–77.

Ellekilde LP and Petersen HG (2013) Motion planning efficient trajectories for industrial bin-picking. *International Journal of Robotics Research* 32(9-10): 991–1004.

Faigl J, Vonásek V and Preucil L (2011) A multi-goal path planning for goal regions in the polygonal domain. In: *Proceedings of IEEE European Conference on Mobile Robots (ECMR)*. pp. 171–176.

Feng S, Xinjilefu X, Atkeson CG and Kim J (2015) Optimization based controller design and implementation for the atlas robot in the darpa robotics challenge finals. In: *Proceedings of IEEE International Conference on Humanoid Robots (Humanoids)*. pp. 1028–1035.

Gäbert C, Kaden S and Thomas U (2021) Generation of human-like arm motions using sampling-based motion planning. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 2534–2541.

Gammell JD, Srinivasa SS and Barfoot TD (2014) Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 2997–3004.

Gammell JD, Srinivasa SS and Barfoot TD (2015) Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 3067–3074.

Gammell JD and Strub MP (2020) A survey of asymptotically optimal sampling-based motion planning methods. *arXiv preprint arXiv:2009.10484* .

Garrett CR, Lozano-Perez T and Kaelbling LP (2018) FFRob: Leveraging symbolic planning for efficient task and motion planning. *International Journal of Robotics Research* 37(1): 104–136.

Google (2017) Solving TSPs with OR-Tools. Available from: `https://developers.google.com/optimization/routing/tsp/tsp`.

Gulletta G, Erlhagen W and Bicho E (2020) Human-like arm motion generation: A review. *Robotics* 9(4): 102.

Hauer F and Tsiotras P (2017) Deformable rapidly-exploring random trees. In: *Proceedings of Robotics: Science and Systems (RSS)*.

Hauser K and Latombe JC (2009) Integrating task and prm motion planning: Dealing with many infeasible motion planning queries. In: *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*.

Janson L, Schmerling E, Clark A and Pavone M (2015) Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *International Journal of Robotics Research* 34(7): 883–921.

Jaquier N, Rozo L, Caldwell DG and Calinon S (2021) Geometry-aware manipulability learning, tracking, and transfer. *International Journal of Robotics Research* 40(2-3): 624–650.

Kalakrishnan M, Chitta S, Theodorou E, Pastor P and Schaal S (2011) STOMP: Stochastic trajectory optimization for motion planning. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 4569–4574.

Karaman S and Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research* 30(7): 846–894.

Kavraki LE, Svestka P, Latombe JC and Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4): 566–580.

Kavraki Lab (2017) The open motion planning library OMPL Available from: `https://ompl.kavrakilab.org/`.

Klein CA and Blaho BE (1987) Dexterity measures for the design and control of kinematically redundant manipulators. *International Journal of Robotics Research* 6(2): 72–83.

Kolakowska E, Smith SF and Kristiansen M (2014) Constraint optimization model of a scheduling problem for a robotic arm in automatic systems. *Robotics and Autonomous Systems* 62(2): 267–280.

Kovács A (2013) Task sequencing for remote laser welding in the automotive industry. In: *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*.

Kovács A (2016) Integrated task sequencing and path planning for robotic remote laser welding. *International Journal of Production Research* 54(4): 1210–1224.

Kuffner JJ and LaValle SM (2000) RRT-connect: An efficient approach to single-query path planning. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 2. pp. 995–1001.

LaValle SM (2006) *Planning Algorithms*. Cambridge University Press.

Lee JJH, Frey K, Fitch R and Sukkarieh S (2014) Fast path planning for precision weeding. In: *Proceedings of Australasian Conference of Robotics and Automation (ACRA)*.

Liarokapis M, Bechlioulis CP, Artemiadis PK and Kyriakopoulos KJ (2017) Deriving humanlike arm hand system poses. *Journal of Mechanisms and Robotics* 9(1): 011012.

Lin YC and Berenson D (2016) Using previous experience for humanoid navigation planning. In: *Proceedings of IEEE International Conference on Humanoid Robots (Humanoids))*. pp. 794–801.

Luna R, Moll M, Badger J and Kavraki LE (2020) A scalable motion planner for high-dimensional kinematic systems. *International Journal of Robotics Research* 39(4): 361–388.

Morrison D, Tow AW, Mctaggart M, Smith R, Kelly-Boxall N, Wade-Mccue S, Erskine J, Grinover R, Gurman A, Hunn T et al. (2018) Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 7757–7764.

Mukadam M, Dong J, Yan X, Dellaert F and Boots B (2018) Continuous-time gaussian process motion planning via probabilistic inference. *International Journal of Robotics Research* 37(11): 1319–1340.

Natarajan S (2021) Learning initial trajectory using sequence-to-sequence approach to warm start an optimization-based motion planner. In: *Proceedings of IEEE/RSJ International conference on intelligent robots and systems (IROS)*.

Noon CE and Bean JC (1993) An efficient transformation of the generalized traveling salesman problem. *INFOR: Information Systems and Operational Research* 31(1): 39–44.

Pan J, Chen Z and Abbeel P (2014) Predicting initialization effectiveness for trajectory optimization. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 5183–5190.

Park C, Pan J and Manocha D (2012) ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In: *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*.

Schulman J, Duan Y, Ho J, Lee A, Awwal I, Bradlow H, Pan J, Patil S, Goldberg K and Abbeel P (2014) Motion planning with sequential convex optimization and convex collision checking. *International Journal of Robotics Research* 33(9): 1251–1270.

Schulman J and the Robot Learning Lab (2013) TrajOpt: trajectory optimization for motion planning. Available from: `https://rll.berkeley.edu/trajopt/doc/sphinx_build/html/`.

Shkolnik A and Tedrake R (2009) Path planning in 1000+ dimensions using a task-space voronoi bias. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 2061–2067.

Silwal A, Davidson JR, Karkee M, Mo C, Zhang Q and Lewis K (2017) Design, integration, and field evaluation of a robotic apple harvester. *Journal of Field Robotics* DOI:10.1002/rob.21715.

Srinivasa SS, Berenson D, Cakmak M, Collet A, Dogar MR, Dragan AD, Knepper RA, Niemueller T, Strabala K, Weghe MV et al. (2012) Herb 2.0: Lessons learned from developing a mobile manipulator for the home. *Proceedings of IEEE* 100(8): 2410–2428.

Srivastava S, Fang E, Riano L, Chitnis R, Russell S and Abbeel P (2014) Combined task and motion planning through an extensible planner-independent interface layer. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 639–646.

Strub MP and Gammell JD (2020) Advanced BIT* (ABIT*): Sampling-based planning with advanced graph-search techniques. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

Sukkar F (2018) *Fast, Reliable and Efficient Database Search Motion Planner (FREDS-MP) for Repetitive Manipulator Tasks*. Master's Thesis, University of Technology Sydney.

Sukkar F, Best G, Yoo C and Fitch R (2019) Multi-robot region-of-interest reconstruction with Dec-MCTS. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 9101–9107.

Tallavajhula A, Choudhury S, Scherer S and Kelly A (2016) List prediction applied to motion planning. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 213–220.

Tompkins A, Senanayake R and Ramos F (2020) Online domain adaptation for occupancy mapping. In: *Proceedings of Robotics: Science and Systems (RSS)*.

Toussaint M (2009) Robot trajectory optimization using approximate inference. In: *Proceedings of International Conference on Machine Learning (ICML)*. pp. 1049–1056.

Toussaint M (2015) Logic-geometric programming: An optimization-based approach to combined task and motion planning. In: *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. pp. 1930–1936.

Wurll C and Henrich D (2001) Point-to-point and multi-goal path planning for industrial robots. *Journal of Robotic Systems* 18(8): 445–461.

You A, Sukkar F, Fitch R, Karkee M and Davidson JR (2020) An efficient planning and control framework for pruning fruit trees. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. pp. 3930–3936.

Yu W (2018) *PID control with intelligent compensation for exoskeleton robots*. Academic Press.

Zhi W, Lai T, Ott L and Ramos F (2021) Trajectory generation in new environments from past experiences. In: *Proceedings of IEEE/RSJ International conference on intelligent robots and systems (IROS)*.

Zucker M, Jun Y, Killen B, Kim TG and Oh P (2013a) Continuous trajectory optimization for autonomous humanoid door opening. In: *Proceedings of IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*. pp. 1–5.

Zucker M, Ratliff N, Dragan AD, Pivtoraiko M, Klingensmith M, Dellin CM, Bagnell JA and Srinivasa SS (2013b) CHOMP: Covariant hamiltonian optimization for motion planning. *International Journal of Robotics Research* 32(9-10): 1164–1193.

Appendix A: Index to multimedia extensions

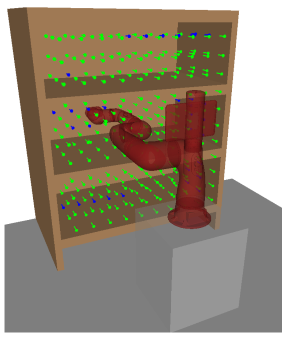
Table of Multimedia Extensions

1	Video	Batch-query experiment with fixed-based manipulator
2	Video	Batch-query experiment with mobile-base manipulator
3	Video	Task-space control experiment

Appendix B: HAP task-space subspace allocations for bookshelf experiments

Here, illustrations of HAP's subspace allocation process are shown. In Fig. B.1, a visualisation of the fixed-base model's naturally extended task-space coverage compared to the mobile-base model is shown. This increased task-space coverage is due to the kinematic configuration of the 7-DOF arm.

In Figs. B.2(a)-(e) each visualisation shows a new subspace found in an iteration of Alg. 4 for the mobile-base model with base mobility disabled. With each iteration, the overall coverage of $\hat{T}$ is increased. Overlap between subspaces in task space is beneficial as it provides additional IK solutions to choose from during online planning. Subspaces found for the mobile-base model in the bookshelf environment and base mobility now enabled are visualised in Figs. B.3(a)-(e) in the order they are generated by HAP. Subspace boundaries are not always obvious and would be difficult to generate manually.


(a) $\hat{T}^0$ for fixed-base model

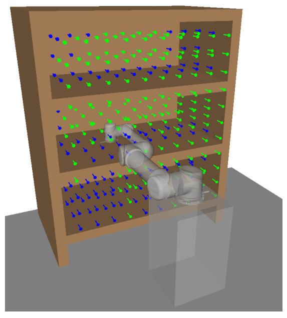

(b) $\hat{T}^0$ for mobile-base model

Figure B.1. Comparison of first subspaces generated for (a) the 7-DOF fixed-base model and (b) the 6-DOF mobile-base model. Green poses indicate areas of task space that lie within the defined subspace; blue poses indicate the remaining unmapped areas. The fixed-base model is capable of almost full task-space coverage with a single subspace, in contrast to the mobile-base model, due to its kinematic redundancy.

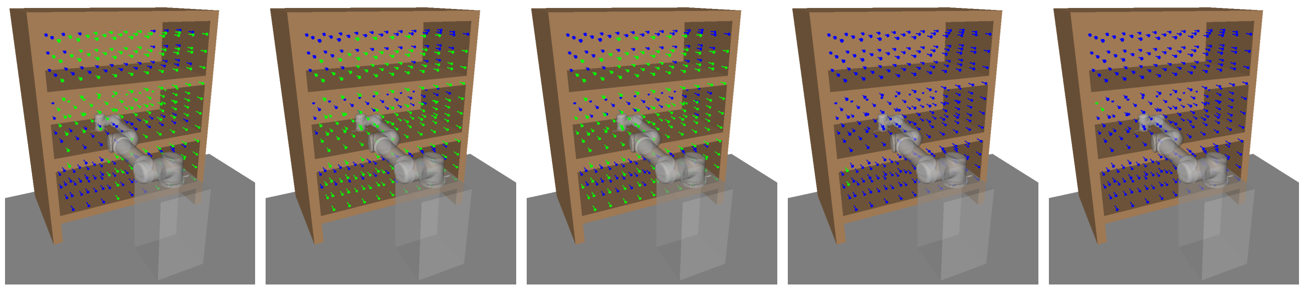

(a) $\hat{T}^0$ **(b)** $\hat{T}^1$ **(c)** $\hat{T}^2$ **(d)** $\hat{T}^3$ **(e)** $\hat{T}^4$

Figure B.2. Visualisation of task-space subspaces for HAP-Multi. Green and blue poses are defined as in the previous figure. Subspaces are sorted by order in which they were found by Alg. 3. Subspaces in (a) and (b) achieve large and diverse coverage, while the subspace in (c) is similar to (b). Subspaces in (d) and (e) cover only small isolated regions of the task space in the left bottom and middle shelf rows, respectively.

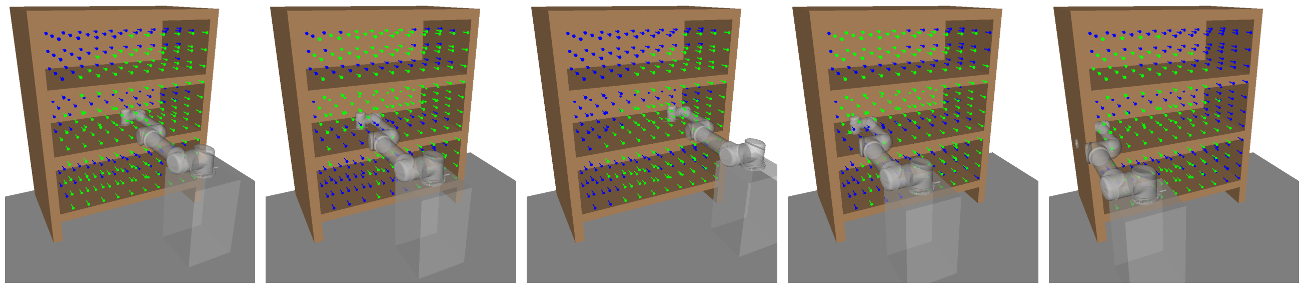
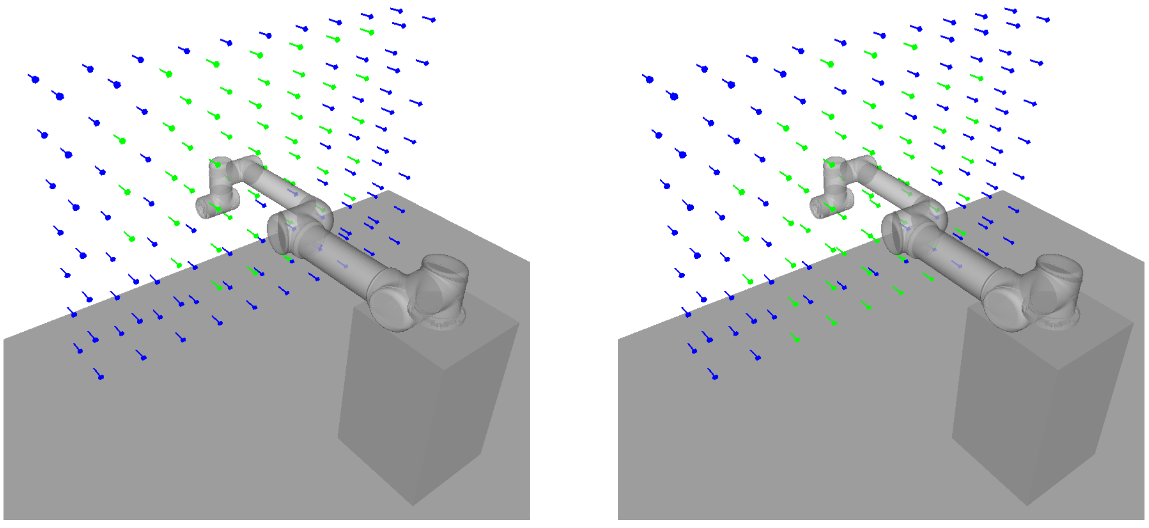

(a) $\hat{T}^0$ **(b)** $\hat{T}^1$ **(c)** $\hat{T}^2$ **(d)** $\hat{T}^3$ **(e)** $\hat{T}^4$

Figure B.3. Visualisation of task-space subspaces for HAP-Mobile. Note the base pose changes for each subspace. Subspaces are sorted by order in which they were found by Alg. 3. Subspaces exhibit large and diverse coverage for all base positions.

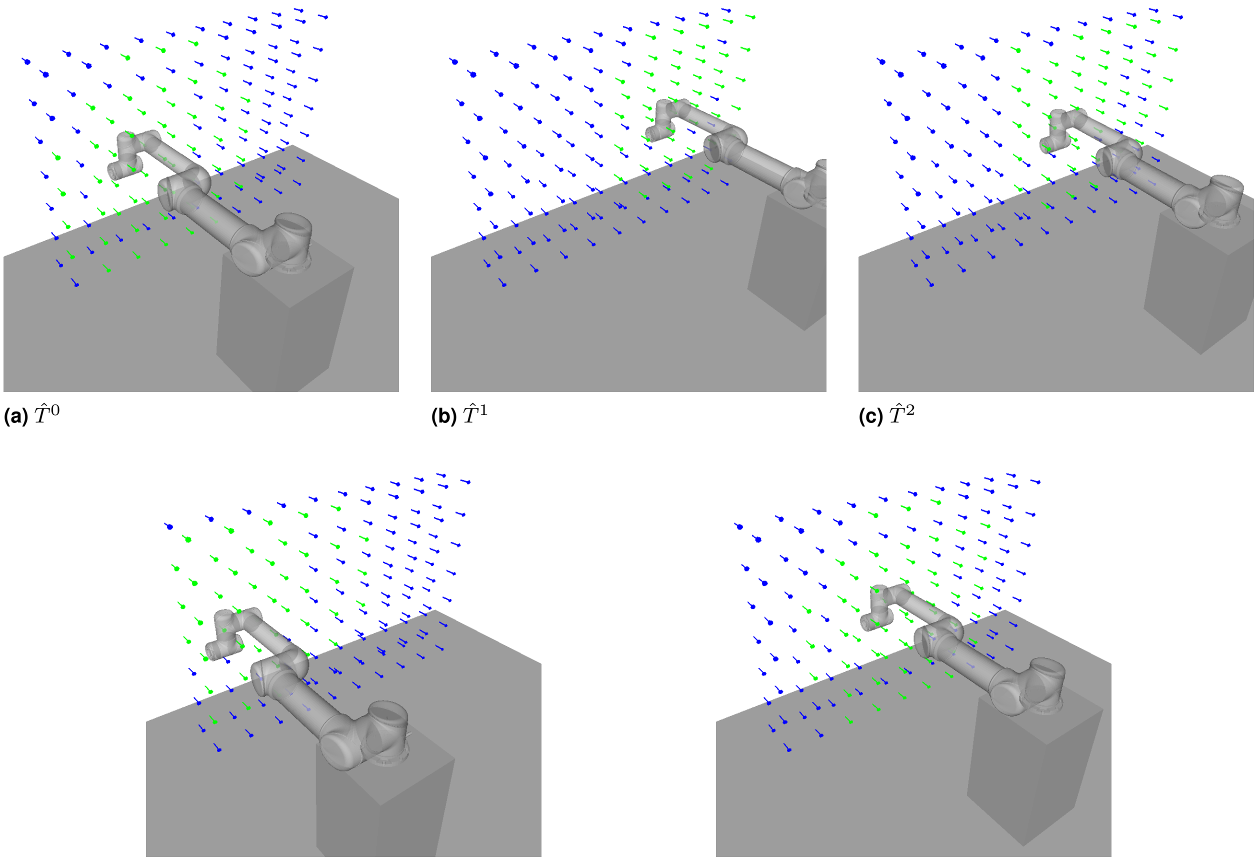
Appendix C: Subspaces in task-space control experiments

Computed subspaces for HAP-Multi and HAP-Mobile are shown in Figures C.1 and C.2, respectively. Allocated subspaces are highlighted in green and their assigned base positions are shown. Unallocated poses in Fig. C.1 are either unreachable from the static base position or have path costs that exceed $c_{\max}$.



(a) $\hat{T}^0$ (b) $\hat{T}^1$

Figure C.1. Subspaces computed for HAP-Multi in the task-space control experiments. The subspace in (a) provides coverage of the back plane of poses while (a) covers the front plane.

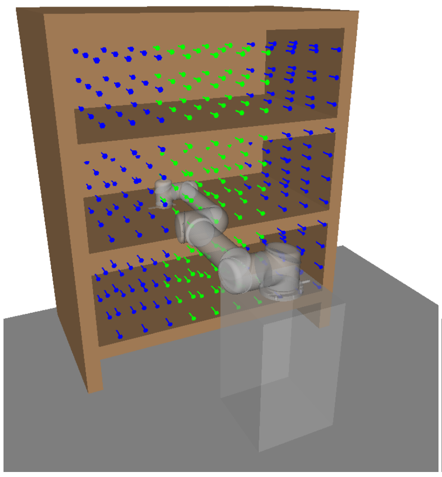


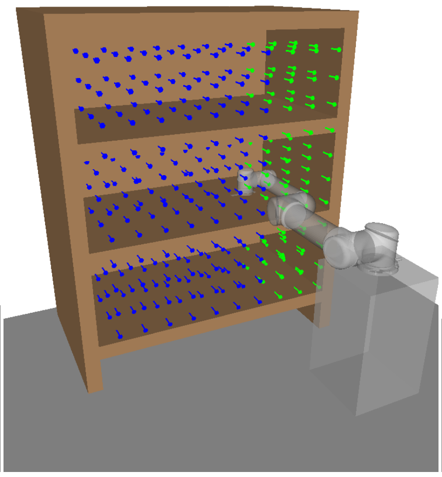
(a) $\hat{T}^0$ (b) $\hat{T}^1$ (c) $\hat{T}^2$ (d) $\hat{T}^3$ (e) $\hat{T}^4$

Figure C.2. Subspaces computed for HAP-Mobile. The five subspaces generated have unique base positions and collectively achieve near-complete coverage of the task space.

Appendix D: Manually generated subspaces and base positions used in baseline methods

Manually defined subspaces for the Naive-Mobile baseline are shown in Fig. D.1 for single- and batch-query experiments and in Fig. D.2 for task-space control experiments. Subspaces areas are indicated in green and their assigned base positions are shown.


(a) Naive allocation 1


(b) Naive allocation 2

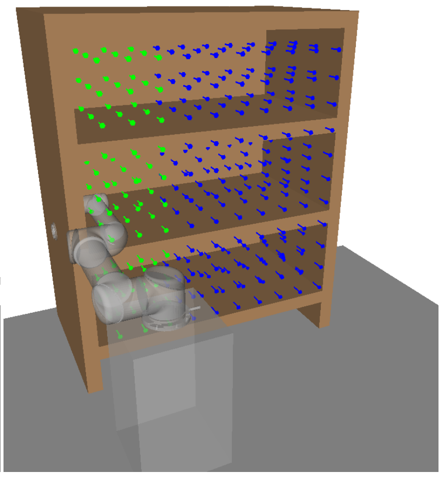
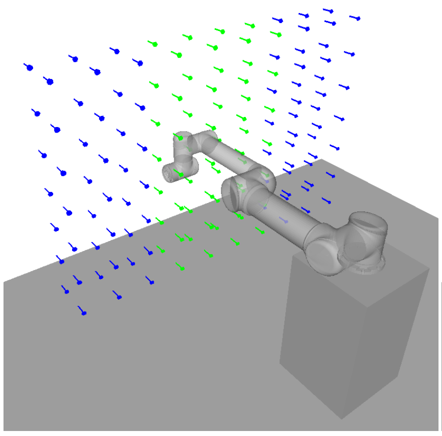
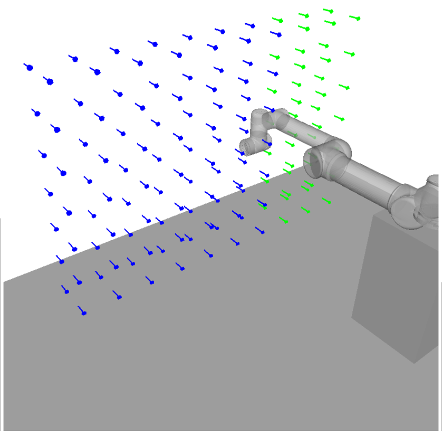

(c) Naive allocation 3

Figure D.1. Subspaces and base positions (manually defined) for Naive-Mobile in single- and batch-query experiments. The task space is divided into three regions with the intent of reducing the reach distance from each of the base positions.


(a) Naive allocation 1


(b) Naive allocation 2

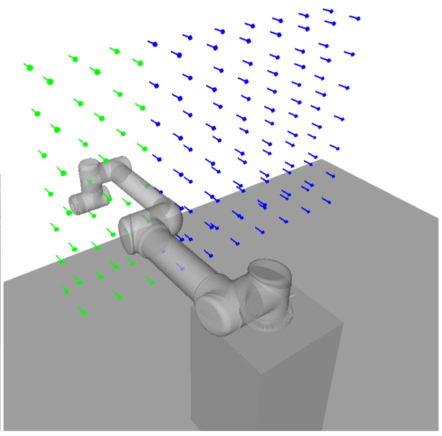

(c) Naive allocation 3

Figure D.2. Subspaces and base positions (manually defined) for the Naive-Mobile method in task-space control experiments. Design intention as in previous figure.