Integrating Motion and Hierarchical Fingertip Grasp Planning

Joshua A. Haustein, Kaiyu Hang and Danica Kragic

***Abstract*— In this work, we present an algorithm that simultaneously searches for a high quality fingertip grasp and a collision-free path for a robot hand-arm system to achieve it. The algorithm combines a bidirectional sampling-based motion planning approach with a hierarchical contact optimization process. Rather than tackling these problems in a decoupled manner, the grasp optimization is guided by the proximity to collision-free configurations explored by the motion planner. We implemented the algorithm for a 13-DoF manipulator and show that it is capable of efficiently planning reachable high quality grasps in cluttered environments. Further, we show that our algorithm outperforms a decoupled integration in terms of planning runtime.**

I. Introduction

Fingertip grasp planning is essential for robotic manipulation, especially for in-hand manipulation. Hence, it has been an active research field in the past decades [1]–[3]. However, due to the high dimensionality of the hand-arm configuration space and the complex objective modeling, contact-level grasp planning and motion planning for a hand-arm system have traditionally been tackled in a strongly decoupled manner. As such, due to kinematic limits or collisions with the environment, a grasp planner generally does not guarantee that a solution can be executed. To this end, recent works have developed systems to integrate grasp and motion planning [4]–[7]. While these works have successfully shown the capability of planning collision-free grasping motions, the grasps are obtained by sampling hand poses in the workspace around the target object followed by an optimization of the reachable grasps given the sampled pose. For this, heuristics such as object dependent superellipsoid work space regions or human trained hand synergies have been used. However, to the best of our knowledge, the problem of simultaneously planning explicit fingertip contacts and an associated motion still remains an open problem.

In this work, we integrate the *Hierarchical Fingertip Space (HFTS)* grasp planner [8] with a bidirectional sampling-based motion planner [9]. The presented algorithm simultaneously explores the robot configuration space as well as the grasp space to compute a path to a feasible high quality grasp. In this process, the grasp search is guided towards feasible solutions through *proximity* to configurations explored in the motion planning search. As illustrated in Fig. 1, our algorithm efficiently detects motion-feasible grasp configuration regions in a hierarchical manner and optimizes promising grasps to construct high quality grasp solutions.

J. A. Haustein, K. Hang and D. Kragic are with the Robotics, Perception and Learning Lab (RPL), CAS, CSC at KTH Royal Institute of Technology, Stockholm, Sweden. {haustein, kaiyuh, dani}@kth.se.

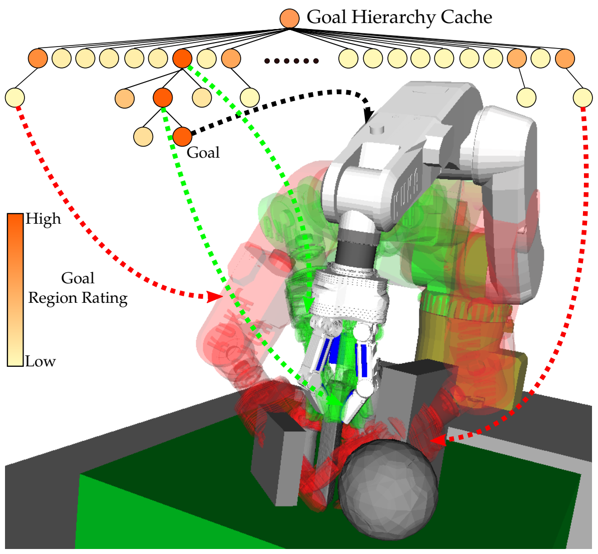

Fig. 1: Our algorithm efficiently explores the hierarchical HFTS search space for feasible high quality grasps while planning hand-arm motions. The exploration is guided by a rating function that measures the feasibility of grasp configurations based on the proximity to collision-free configurations explored in the motion planning process. The figure illustrates the state of exploration after a feasible grasp was found. The transparent configurations (red: in collision, green: collision-free) are associated with the different nodes in the grasp search. **Video**: `http://www.csc.kth.se/~haustein/videos/ICRA2017.mp4`

II. Terminology & Problem Definition

In this work, we address the problem of motion and fingertip grasp planning for a robot with a d_a-DoF arm and a d_h-DoF dexterous hand. We assume that the geometry, the kinematics and state of the robot, the geometry of its environment, as well as the geometry and pose of the target object is known.

For our robot, let $\mathcal{C}$ denote the combined $d = d_a + d_h$ dimensional configuration space and accordingly $\mathcal{C}_{free} \subset \mathcal{C}$ the collision-free subspace. Given a start configuration $\phi_s \in \mathcal{C}_{free}$, our goal is to find a continuous path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = \phi_s$ and $\tau(1) = \phi_g$, where ϕ_g is a goal configuration that lies in a goal region $\mathcal{C}_G \subset \mathcal{C}_{free}$. Our goal is to plan a path to a configuration in $\mathcal{C}_{FC}$, the set of configurations in which the robot achieves a high quality fingertip grasp on the target object. However, formally such configurations are by definition not within $\mathcal{C}_{free}$ and hence we define the goal region as $\mathcal{C}_G = \{\phi \in \mathcal{C}_{free} \mid c(\phi) \in \mathcal{C}_{FC}\}$, where $c(\phi) : \mathcal{C} \rightarrow \mathcal{C}$ closes the hand within some predefined threshold.

A. Fingertip Grasp Planning utilizing HFTS

As described in [10], given a grasp quality function and a robotic hand, contact-level grasps on arbitrary object shapes

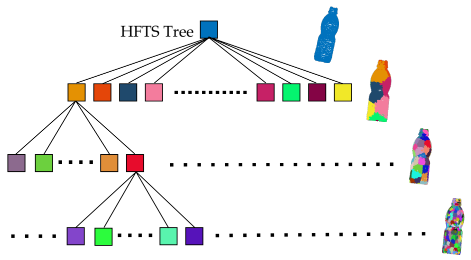

Fig. 2: The HFTS tree for grasp planning. The nodes on different levels represent approximate grasps with different partitioning resolutions.

can be planned efficiently using the *Hierarchical Fingertip Space (HFTS)*. In brief the HFTS is constructed as follows. Given a point cloud of an object's surface, a discrete set of candidate contacts is extracted by using a user-defined filtering function to filter out unsuitable contacts. Thereafter, this set is recursively partitioned into subsets of contacts to approximate contacts of different resolution on each level. These partitions are then used to construct the Hierarchical Fingertip Space, denoted H. Concretely, for m-contact grasping, on each partitioning level a candidate grasp h is formed as an m-tuple of partitions. Hence, each level of the hierarchy consists of all potential combinations of partitions on that level. As we descend in the hierarchy, the nodes approximate more accurate candidate grasps as shown in Fig. 2. A grasp h_{i+1} on level $i+1$ is a child of grasp h_i if and only if all contact partitions of h_{i+1} are respectively subsets of h_i's contact partitions.

Since Ferrari-Canny's contact-level grasp quality function [11] is Lipschitz continuous [12], we can approximate the grasp quality $Q(h) \in \mathbb{R}$ of lower level grasps by the quality of their higher level ancestors. For this, we compute the quality of a higher level grasp, which is an m-tuple of contact partitions, as the quality of the representative grasp made of the mean positions and normals in its m contact partitions. In addition to Q, [10] evaluates a reachability function $R(h) \in \mathbb{R}^{\geq 0}$ of a robotic hand for a candidate grasp h to linearly relax the hand reachability constraint. Here, a smaller $R(h)$ indicates better reachability.

A stable and reachable grasp can be obtained in the HFTS by optimizing

$$\max_{h \in H} \frac{Q(h)}{R(h) + \alpha} \tag{1}$$

where $\alpha \in \mathbb{R}^{+}$ is a weighting factor to prevent the division by zero in case h is exactly reachable. Starting from the top of the HFTS, the algorithm maximizes Eq. (1) to pursue better grasp quality and reachability. The maximization is conducted by stochastic hill climbing with a fixed number of iterations on each level. After a solution $\hat{h}_i$ is found on level i, the algorithm descends to level $i+1$ and continues a finer optimization on the children of $\hat{h}_i$. This procedure is continued until a grasp is found on the bottom level. Thereafter, our previous work [10] first computes a hand configuration by querying a precomputed database followed by computing an arm configuration for posing the hand.

B. Problem Definition

The optimization procedure as described above considers neither the kinematic restrictions of the arm nor collisions with the environment. As a consequence, many of the computed grasps are not feasible for execution. Thus a naive integration of the grasp planner into a motion planner as goal sampler results in a significant amount of computational effort being wasted on computing infeasible grasps.

Hence, instead of solving Eq. (1), the grasp planner should ideally solve

$$\begin{aligned} \underset{h \in H}{\text{maximize}} \quad & \frac{Q(h)}{R(h) + \alpha} \\ \text{subject to} \quad & \Phi(h) \cap \mathcal{C}_{free} \neq \emptyset \\ & \exists \tau \in \Xi : \tau(0) = \phi_s \wedge \tau(1) \in \Phi(h) \end{aligned} \tag{2}$$

where $\Phi : H \rightarrow 2^{\mathcal{C}}$ maps fingertip grasps to the set of robot configurations achieving these and $\Xi = \mathcal{C}_{free}^{[0,1]}$ is the set of collision-free paths. We refer to the first constraint as reachability constraint and to the second constraint as connectability constraint. Note that fulfilling connectability implies fulfilling reachability. However, reachability is generally easier to relax, which is why we distinguish between the two.

Various precomputed heuristics have been proposed that would allow a relaxation of the reachability constraint in environments with few obstacles [13]–[15]. In the presence of many obstacles, however, these methods do not suffice as they are generally difficult to adjust to collisions. The connectability constraint is a difficult problem because in order to determine whether a path between two configurations exists, an algorithm needs to determine whether both configurations lie within the same connected component of $\mathcal{C}_{free}$. Therefore, rather than computing for each $h \in H$ explicitly whether both constraints are fulfilled, the key idea of this work is to utilize the knowledge about $\mathcal{C}_{free}$ that is gradually acquired online by a motion planning algorithm to estimate which regions of H are likely to fulfill the constraints. For this, our underlying assumption is that grasps with similar contacts can be achieved by similar robot configurations. Thus we assume that if a grasp h_i on level i fulfills the aforementioned constraints, the descendants of h_i are also likely to fulfill the constraints.

III. Methodology

An overview of our framework is shown in Fig. 3. The framework consists of two separate components that share current knowledge of $\mathcal{C}_{free}$. On one side, a modified version of the Constrained Bi-directional Rapidly-Exploring Random Tree (CBiRRT) algorithm [16] explores $\mathcal{C}$ and attempts to connect goal configurations to the start configuration. On the other side, a goal sampling algorithm utilizes the HFTS fingertip grasp planner to search for these goal configurations.

The key idea behind our framework is to guide the goal sampling towards regions of the grasp search space that appear to be promising to fulfill the reachability and connectability constraint. This guidance is performed based on the current knowledge the system has acquired on $\mathcal{C}_{free}$. As the motion planning algorithm is constructing search trees, knowledge on $\mathcal{C}_{free}$ is gained in terms of samples.

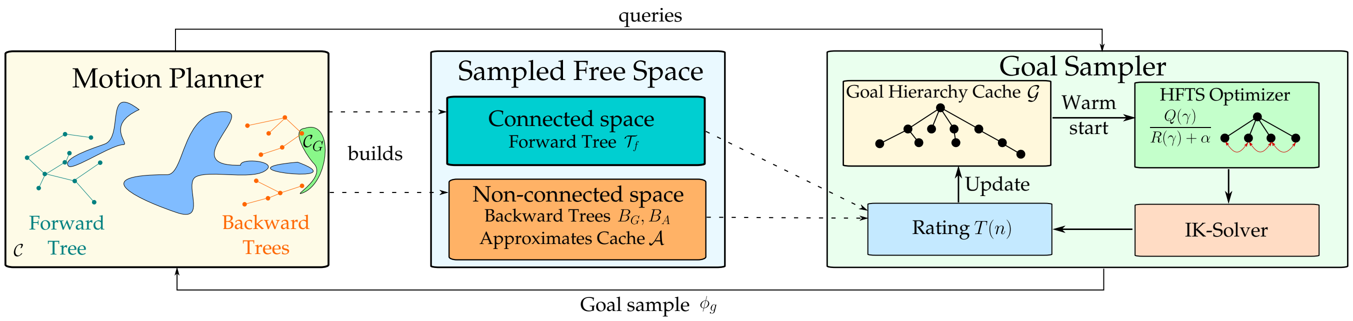

Fig. 3: Our framework consists of two components. A bidirectional sampling-based motion planning algorithm is used to explore the robot configuration space to find a path between the start configuration and a reachable fingertip grasp. The goal sampler, in turn, provides configurations that achieve high quality fingertip grasps on the target object. For this, the goal sampler relies on the current knowledge of $\mathcal{C}_{free}$, which is shared with the motion planning algorithm.

These samples can be classified in two sets: connected free-space and non-connected free-space. The connected free-space is the set of samples that are connected to the start configuration, i.e. the forward tree $\mathcal{T}_f$. The non-connected free-space is the set of samples that is known to be collision-free, but not connected to the forward tree. This set consists of the backward trees that are constructed by the motion planner as well as a set $\mathcal{A} \subset \mathcal{C}_{free}$ that is maintained by the goal sampler.

When the motion planner queries the goal sampler for a new goal it, is not guaranteed that it succeeds in computing a valid goal. In this case, the goal sampler provides the motion planner with an approximate goal $\phi_g \in \mathcal{A}$, if available. An approximate goal is a collision-free configuration that is likely to be in some neighborhood to some valid goal configuration. The motivation for this is twofold. First, it provides the motion planner with configurations that drive the exploration into relevant regions of $\mathcal{C}_{free}$. Second, if such an approximate configuration is connected to $\mathcal{T}_f$, the goal sampler gains an improved estimation on the connectability constraint. As a consequence, however, we need to distinguish between different types of backward trees - those that root in goal and those that root in approximate goal configurations. We denote these two sets as B_G and B_A respectively.

A. *Motion Planning*

Rapidly-Exploring Random Tree (RRT) based algorithms have shown to be successful even in high dimensional search spaces and are commonly applied to a variety of search problems [9], [17]–[19]. The CBiRRT algorithm at hand builds such search trees incrementally from both start and goal configurations, while respecting constraints on the configuration space. Our modification of the algorithm is shown in Algorithm 1. In contrast to the original algorithm from [16], our modified version maintains the two sets of backward trees B_G and B_A. In each iteration the algorithm either extends the existing trees and attempts to connect a backward tree to the forward tree, or samples a new goal configuration. The sampling of a goal configuration is performed by the goal sampling algorithm shown in Algorithm 2 and is explained in detail in III-B.

The probability p_g, by which a new goal configuration is sampled, is dynamically adjusted based on the number of backward trees currently available. This dynamic adjustment allows the algorithm to first focus on finding (approximate) goal configurations. Once backward trees exist, p_g decreases allowing the algorithm to focus on extending the search trees rather than goal sampling. Since it is not known whether any of the backward trees lie within the same connected component as the forward tree, p_g does not decrease to 0 but instead to some minimal goal sampling probability p_{min}.

When the algorithm succeeds in connecting a backward tree $\mathcal{T}_b$ to the forward tree, we need to distinguish between $\mathcal{T}_b$ being rooted at an approximate goal or a real goal. In case it originates from an approximate goal, i.e. $\mathcal{T}_b \in B_A$, $\mathcal{T}_b$ is merged with the forward tree and the search continues. Note that in this case, the goal sampling probability p_g dynamically increases again. Furthermore, by merging $\mathcal{T}_b$ with $\mathcal{T}_f$ all configurations stored in $\mathcal{T}_b$ become part of the connected free-space, which, in turn, influences the goal sampling algorithm.

The algorithm terminates when either the total processing time exceeds some user defined threshold or the algorithm connects the forward tree to a backward tree originating from a real goal, i.e. $\mathcal{T}_b \in B_G$. In the latter case the path connecting $\phi_s \in \mathcal{T}_f$ and $\phi_g \in \mathcal{T}_b$ is extracted and smoothed by a shortcut algorithm similar to [20].

1) Extending Search Trees: The tree extension procedure alternates between a *forward* and *backward* direction. In the *forward* direction it first extends the forward tree and then a backward tree; in the *backward* direction it is vice versa. The extension for a tree $\mathcal{T}$ is performed as it is common for RRTs by sampling a random configuration $\phi_r \in \mathcal{C}_{free}$ followed by an attempt to connect the closest configuration in $\mathcal{T}$ to ϕ_r. Thereafter, an attempt is made to connect the forward tree to a backward tree.

In case the search direction is *forward*, the forward tree is first extended towards the random sample. Then, the backward tree that is closest to the new forward tree node is selected for an connection attempt. In case the search direction is *backward* a random backward tree is selected and extended towards the random sample. Then, similar as before, a connection attempt between the selected backward tree and the forward tree is made. Selecting the backward tree randomly guarantees that no backward tree overshadows the expansion of other backward trees. Since connecting back-

Algorithm 1: The motion planning algorithm.

```
Input: Start configuration $\phi_s \in \mathcal{C}_{free}$, Time limit $T_{max}$, Goal hierarchy
       search space $H$
Constants: Goal probability parameters $p_{max}, p_w, p_{min}$
Output: Path $[\phi_s, \ldots, \phi_g]$ or $\emptyset$
$\mathcal{T}_f \leftarrow$ TREE($\phi_s$)
dir $\leftarrow$ forward
$B_G \leftarrow \emptyset$; $B_A \leftarrow \emptyset$
$\mathcal{G} \leftarrow$ CREATEGOALHIERARCHYCACHE($H$)
$\mathcal{A} \leftarrow \emptyset$
while not TERMINALCONDITION($T_{max}$) do
    $p \leftarrow$ SAMPLEUNIFORM([0, 1])
    $p_g \leftarrow p_{max} e^{-p_w |B_G \cup B_A|} + p_{min}$
    if $p \leq p_g$ then
        $\phi_g \leftarrow$ SAMPLEGOAL($\mathcal{T}_f$, $B_G \cup B_A$, $\mathcal{A}$, $\mathcal{G}$)
        if $\phi_g \in \mathcal{C}_{goal}$ then
            $B_G \leftarrow B_G \cup \{$TREE($\phi_g$)$\}$
        else if $\phi_g \in \mathcal{C}_{free}$ then
            $B_A \leftarrow B_A \cup \{$TREE($\phi_g$)$\}$
    else
        bConnected, dir, $\mathcal{T}_b \leftarrow$ EXTENDTREES($\mathcal{T}_f$, $B_G$, $B_A$, dir)
        if bConnected then
            $\mathcal{T}_f \leftarrow \mathcal{T}_f \cup \mathcal{T}_b$
            if $\mathcal{T}_b \in B_G$ then
                return EXTRACTANDSHORTCUTPATH($\mathcal{T}_f$)
            else
                $B_A \leftarrow B_A \setminus \{\mathcal{T}_b\}$
return $\emptyset$
```

ward trees that originate from real goals is more desirable, we bias the random selection towards these backward trees with probability $p_{goalTree}$.

2) Projection Heuristic for Backward Tree: For the extension of a search tree, we utilize the *ConstrainedExtend* method as presented in [16]. This method extends a tree iteratively from a given start towards a target configuration with some step size $\Delta\phi$. In this process it utilizes a projection function to project intermediate configurations to a constraint manifold. In this work, we do not define a constraint manifold. Instead, we utilize a projection function to assist the motion planner in extending backward trees from grasping configurations. These configurations are by definition close to collisions and generally any path leading to one must pass through narrow passages. Hence, a tree growth guided by uniform random sampling is likely to fail in many cases.

Given a previously sampled configuration $\phi_{old} \in \mathcal{C}_{free}$ and a newly sampled configuration $\phi_{new} = \phi_{old} + \Delta\phi$ our projection function attempts to project ϕ_{new} to $\mathcal{C}_{free}$ if necessary. For this, the method first determines whether the end-effector in configuration ϕ_{old} is within some user-defined distance D_m to the target object and whether $\phi_{new} \notin \mathcal{C}_{free}$. Only if these conditions are fulfilled, the method computes a heuristic configuration gradient that moves the end-effector away from the target position and opens the hand. The projected configuration is then $\phi'_{new} = \phi_{old} + \langle g, h\rangle h$, where $g = \phi_{new} - \phi_{old}$ is the intended direction of movement and h the heuristic gradient computed. In case the aforementioned conditions are not fulfilled or $\langle g, h\rangle \leq 0$, the projection function is the identity function. Therefore, it does not limit the approach directions. Furthermore, this projection is only applied when extending backward trees.

3) Distance Function: The performance of any RRT-based algorithm is strongly influenced by the distance function on $\mathcal{C}$. For $\phi, \psi \in \mathcal{C}$ we utilize the norm

$$\|\phi - \psi\| = sqrt(\sum_{i=1}^{d} \omega_i(\phi_i - \psi_i)^2) \tag{3}$$

where $\omega_i \in \mathbb{R}^{\geq 0}$ are weights for each DoF.

Algorithm 2: The goal sampling algorithm SAMPLEGOAL.

```
Input: Forward tree $\mathcal{T}_f$, Set of backward trees $\mathcal{B}$, Cache of approximate goals
       $\mathcal{A}$, Goal hierarchy cache $\mathcal{G}$,
Output: A robot configuration $\phi \in \mathcal{C}_{free}$ that is either a goal or close to a
        goal. $\perp$ if no new $\phi \in \mathcal{C}_{free}$ was found.
Constants: Connected free space weight $w_c$, Non-connected free space weight
           $w_f$, Number of samples $\kappa$, Number of iterations $i_{min}, i_{max}$
$n_p \leftarrow$ ROOT($\mathcal{G}$)
$k \leftarrow \kappa$
while $k > 0$ do
    $n_c \leftarrow$ PICKCHILD($n_p$)
    if SHOULDDESCEND($n_p$, $n_c$) then
        $n_p \leftarrow n_c$
    else if $cvr(n_p) = 1$ then
        // If all children have been sampled
        if $\mathcal{C}_{n_c} \cap \mathcal{C}_{goal} \neq \emptyset$ then
            $\phi_g \leftarrow$ SAMPLENULLSPACE($n_c$)
            if $\phi_g \in \mathcal{C}_{free}$ then
                return $\phi_g$
        $k \leftarrow k - 1$
    else
        // Sample a new child
        $i \leftarrow i_{min} + \frac{T(n_p)}{w_f + w_c}(i_{max} - i_{min})$
        $n_c \leftarrow$ SAMPLENEWCHILD($n_p$, $i$)
        // Implicitly adds $n_c$ to $\mathcal{G}$
        $Ch_a(n_p) \leftarrow Ch_a(n_p) \cup \{n_c\}$ // See Sec. III-B.6.
        $Ch(n_p) \leftarrow Ch(n_p) \cup \{n_c\}$
        $\{\phi\} \leftarrow \mathcal{C}_{n_c}$
        if $\phi \in \mathcal{C}_{goal}$ then
            // Implies $n_c$ is a leaf.
            return $\phi$
        if $\phi \in \mathcal{C}_{free}$ then
            $\mathcal{A} \leftarrow \mathcal{A} \cup \{\phi\}$
        $k \leftarrow k - 1$
if $|\mathcal{A}| > 0$ then
    $\phi \leftarrow$ SAMPLEUNIFORM($\mathcal{A}$)
    $\mathcal{A} \leftarrow \mathcal{A} \setminus \{\phi\}$
    return $\phi$
return $\perp$
```

B. $\mathcal{C}_{free}$-Proximity-Guided Goal Sampling

The goal sampling algorithm shown in Algorithm 2 utilizes the HFTS grasp search space to search for a new grasp. Rather than descending all the way to the bottom of the HFTS hierarchy as described in II-A, the algorithm only optimizes Eq. (1) for one level at a time. On each level the algorithm evaluates whether to descend further in the hierarchy or to keep searching for alternative local optima based on a heuristic T for the reachability and connectability constraints, which is defined later in Eq. (5).

1) The Goal Hierarchy Cache $\mathcal{G}$: For this the algorithm incrementally builds a tree $\mathcal{G}$, denoted goal hierarchy cache, that stores the local optima found on each level, as illustrated in Fig. 1. Each node $n \in \mathcal{G}$ in this structure is associated with an HFTS node $\eta(n) \in H$. The hierarchical structure of H is transferred to $\mathcal{G}$ in the sense that the parent of a node $p(n) \in \mathcal{G}$ is the node that is associated with the parent of $\eta(n)$. Furthermore, a node n stores all computed robot configurations $\mathcal{C}_n$ for the grasp $\eta(n)$.

Algorithm 3: SHOULDDESCEND

```
Input: Hierarchy node $n_p$, Child node $n_c$
Output: Decision whether to descend or not.
Constants: Weights $w_f, w_c$
1 if $n_c = \perp$ or $n_c$ is leaf then
2     return False
3 if $cvr(n_p) = 1$ then
4     return True
5 $p \leftarrow$ SAMPLEUNIFORM($[0, 1]$)
6 if $p \leq \frac{T_p(n_p)}{T_p(n_p)+T_c(n_c)}$ then
7     return False
8 return True
```

Algorithm 4: PICKCHILD

```
Input: Hierarchy node $n_p$
Output: Child node $n_c$
Constants: Weights $w_f, w_c$, Maximum number of active children $N_a$
1 if $|Ch(n_p)| = 0$ then
2     return $\perp$
  // Re-activate an old child
3 $Ch_a(n_p) \leftarrow Ch_a(n_p) \cup$ SAMPLEUNIFORM($Ch(n_p)$)
4 while $|Ch_a(n_p)| > N_a$ do
5     $n_r \leftarrow$ SAMPLEWEIGHTED($Ch_a(n_p)$, $\frac{1}{T_c}$)
6     $Ch_a(n_p) \leftarrow Ch_a(n_p) \setminus \{n_r\}$
  // Sample an active child with priority to promising
     children
7 return SAMPLEWEIGHTED($Ch_a(n_p), T_c$)
```

2) Sampling procedure: With this hierarchy, the goal sampling algorithm proceeds as shown in Algorithm 2. Each search for a new goal starts at the root of $\mathcal{G}$, which is associated with the root of H and is the only initial node in $\mathcal{G}$. Starting from the root the algorithm proceeds iteratively to either descend towards a previously cached node, sample the null space for the grasp of the current node n_p or sample a new child node. In each iteration the PICKCHILD function, Algorithm 4, queries the cache for a child node from the set of n_p's cached children $Ch(n_p)$. If a child $n_c \in Ch(n_p)$ exists in $\mathcal{G}$, the function SHOULDDESCEND, Algorithm 3, decides whether the algorithm should descend to n_c or stay at n_p and sample a new child by calling the function SAMPLENEWCHILD. If there is no cached child in $\mathcal{G}$, i.e. $Ch(n_p) = \emptyset$, the algorithm directly samples a new child. If there are no new children left to sample, the null space for $\eta(n_p)$ is sampled.

3) Sampling a New Child: SAMPLENEWCHILD(n_p, i) samples a new goal hierarchy node by performing local stochastic optimization of Eq. (1) on the children of $\eta(n_p)$ that have not been sampled before, i.e. $\{h \in H \mid h \in \delta(\eta(n_p)) \wedge \nexists\, n \in Ch(n_p) : h = \eta(n)\}$, where $\delta(h)$ denotes the set of children of h. The number of iterations i for the optimization depends on the rating $T(n_p)$ the parent node achieves. Thus the algorithm spends more computational effort on promising than on non-promising grasps.

For the resulting approximate local optimum h^* an IK solver is used to search for a robot configuration that at most collides with the target object at the contacts associated with the grasp. If such a configuration does not exist due to environment collisions, a colliding configuration is computed instead. In either cases, a new cache node n_c is created that stores the found configuration in a set $\mathcal{C}_{n_c}$. If there is no kinematic solution, the node is still created, but no configuration is stored, i.e. $\mathcal{C}_{n_c} = \emptyset$. In order to allow the motion planner to connect to configurations in contact, we modify these by opening the hand by a small angle. Thereafter, the newly created cache node is saved in the cache hierarchy $\mathcal{G}$ as child of n_p. If $h^* = \eta(n_c)$ is a leaf, the configuration computed for h^* is collision-free, and h^* is a stable grasp, the configuration is a goal and returned. Otherwise, if the configuration is collision-free, we store it as an approximate goal.

If no goal configuration is found, the algorithm terminates after κ new configurations were sampled. In our experiments we choose κ slightly larger than the depth of H, which allows the algorithm to descend to the lowest level of the hierarchy, while sampling some alternatives. In case the search for a goal was unsuccessful, an approximate goal is returned.

4) Rating Function t(n): The main purpose of this algorithm is to guide the grasp search towards regions of H that are promising to fulfill the reachability and connectability constraint. For this, we rate each node n that is stored in $\mathcal{G}$ by a rating function

$$t(n) = \begin{cases} \max\limits_{\phi \in \mathcal{C}_n} w_f e^{-\Omega(\phi)} + w_c e^{-\Gamma(\phi)} & \text{if } \mathcal{C}_n \neq \emptyset \\ cvr(p(n))t(p(n)) & \text{if } \mathcal{C}_n = \emptyset \\ w_f & \text{if } n \text{ is root} \end{cases}, \quad (4)$$

where $w_c, w_f \in \mathbb{R}^{>0}$ are weights and $\Omega(\phi)$ the shortest distance of ϕ to the known non-connected and $\Gamma(\phi)$ the shortest distance to the connected free-space. Hence, the closer a configuration of a node is to either the non-connected free-space or the connected free-space, the larger the rating.

In case there is no configuration available for a cached node n, we assign the rating of the parent $p(n)$ weighted by the parent's coverage $cvr(n)$. The coverage for a node n is the ratio between the number of cached children of the node and the maximum number of children possible. Hence, for a node without any configuration the rating is initially very low and increases as more and more of its siblings are sampled. The rating for the root node is constant and by definition as high as the minimal rating of a collision-free node.

Each time the goal sampling algorithm searches for new goals, $\mathcal{G}$ grows and more information on H is available from previous explorations. Therefore, rather than basing the rating of a cached node n purely on its own configurations, we choose to compute a branch rating

$$T(n) = \frac{1}{2}(t(n) + \frac{1}{|Ch_a(n)|} \sum_{n' \in Ch_a(n)} T(n')) \quad (5)$$

that takes the ratings of the cached descendants of n into account. $Ch_a(n)$ denotes the set of active children of n, which are explained in Sec. III-B.6.

5) Decision on Descent: Given this rating function, the function SHOULDDESCEND(n_p, n_c) determines whether the algorithm should descend from a node n_p to n_c or remain at n_p and sample a new child. This decision algorithm is shown in Algorithm 3. In case the child is a leaf node or there is no cached child available, no descent is made. If in the other extreme all possible children of n_p have been sampled, i.e. $cvr(n_p) = 1$, the algorithm descends without further checks.

Otherwise the choice is randomized with the probability of remaining at the same level being proportional to the rating $T_p(n) = (1 - cvr(n))T(n)$. On the other hand, the probability of descending to the child node is proportional to $T_c(n) = (1 - \frac{|L(n)|}{|\widehat{L(n)}|+1})T(n)$, where $|L(n)|$ denotes the number of cached leaves in the branch rooting at n and $|\widehat{L(n)}|$ the maximum number of leaves the branch can have.

The definitions of $T_c(n)$ and $T_p(n)$ aim at balancing between exploring new regions of H and searching for new grasps in known promising regions. The probability of remaining at a node is large, if the rating of the node n is good and there are many new children left to explore. If on the other hand, most of the children have already been explored, the probability decreases as it is unlikely to find more high quality subbranches. The probability of descending to a child, in turn, is large if the child achieves a high rating and there are many unsampled leaves in the branch left. The more the child branch has been explored, the smaller the probability of descending becomes. Note that $T_c(n) > 0$ at all times, which allows the algorithm to eventually explore any node if it does not terminate before.

6) Picking a Child: Similar to the decision on descending, the selection of a candidate child to descend to is made based on the ratings of the children. The accuracy of our rating function depends on the degree of exploration of $\mathcal{C}_{free}$. Hence, in early stages, it may assign low ratings to branches that contain feasible grasps. We therefore need to ensure there remains a chance of exploring branches with low ratings. Thus we randomize the procedure, as shown in Algorithm 4. The probability of a child being selected is proportional to its rating T_c, i.e. $p(n_c) = \frac{T_c(n_c)}{\sum_{c \in Ch_a(n_p)} T_c(n_c)}$. However, as the number of cached children of a node n_p increases, high ratings become less significant as we need to normalize by the summed ratings of all children. As a consequence, the chance of the algorithm to descend towards promising children would decrease the more children a node has. To compensate for this we distinguish between active children $Ch_a(n_p)$ and inactive children $Ch(n_p) \setminus Ch_a(n_p)$. $Ch_a(n_p)$ is a limited set of children that is updated every time the function PICKCHILD is executed by first adding a random inactive child and then down sampling the set until it has at most N_a elements. The probability of a child n_c being removed from the set of active children is anti-proportional to its rating $T_c(n_c)$. Note also that in Algorithm 2 line 16 any newly sampled child is initially added as an active child.

IV. EXPERIMENTS

We implemented the proposed system in Python using the OpenRAVE simulation environment [21]. Our virtual robot is a model of the KUKA KR5 sixx 850 arm with $d_a = 6$ DoFs in combination with a Schunk-SDH hand with $d_h = 7$ DoFs, making a total dimension of $d = 13$. The Schunk-SDH hand has three fingers and accordingly we plan 3-contact fingertip grasps on the Hierarchical Fingertip Space. All experiments

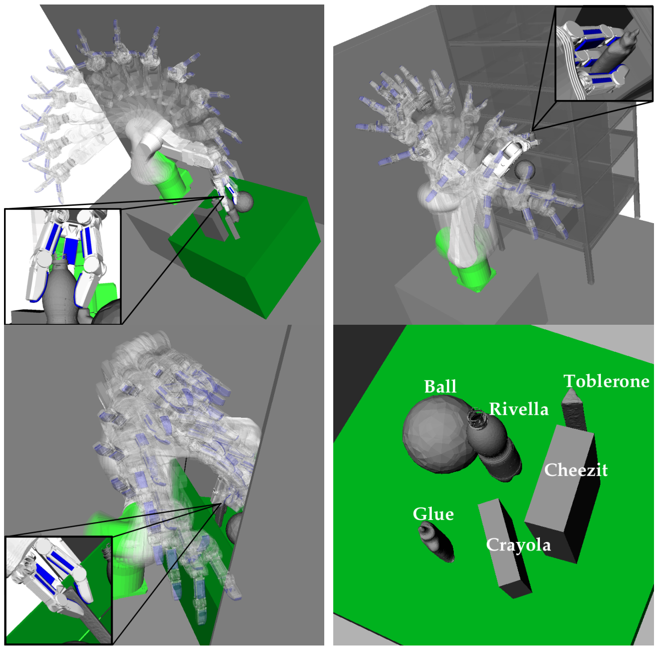

Fig. 4: The three test environments, the different test objects and example grasps with approach paths.

were run on a machine with an Intel Core i7-4790K CPU @ 4.00GHz×4 and 16GB RAM running Ubuntu 14.04. The parameters used in our experiments are shown in Fig. 6.

We evaluate the presented algorithm in the three different environments shown in Fig. 4 for different object poses. In each environment the robot is tasked with grasping various objects of different sizes and shapes that are surrounded by obstacles. Following the definition in [8], the HFTSs of all objects have the structure $20-4-4$ resulting in respectively $20^3, 4^3, 4^3$ number of children per node on each level. As can be seen in Fig. 4 as well as in the accompanying video, our algorithm successfully computes fingertip grasps as well as motion plans towards these.

In order to evaluate the performance of our algorithm, we compare it to a decoupled integration of the HFTS fingertip grasp planner. In this decoupled version we replace our goal sampling algorithm with the algorithm described in Sec. II-A, which simply optimizes $Eq.\ (1)$ without considering the reachability nor the connectability constraint.

Fig. 5 shows the explored HFTS subspace for both algorithms for two typical planning instances on the same test case. As it can be seen, the decoupled version is generally required to explore a significantly larger portion of the search space until it finds a grasp for which a motion plan is found. In fact, in many planning instances the decoupled algorithm only finds very few different grasps. In contrast, our coupled algorithm typically explores a significantly smaller portion of the HFTS space and finds more feasible solutions. As it can be seen in Fig. 5 our algorithm is more conservative than the decoupled version and only descends in the hierarchy, if the rating function indicates a branch is promising.

As a consequence we expect our algorithm to be in average faster than the decoupled version. In order to evaluate this, we execute both algorithms on all environments 150 times per object with a time limit of $60s$. Our algorithm

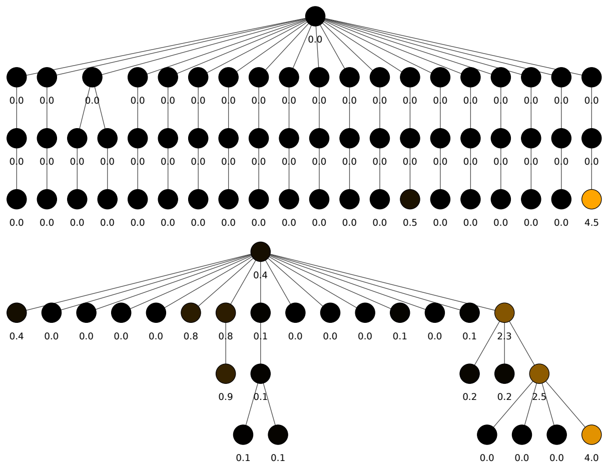

Fig. 5: The explored HFTS of an example execution of the decoupled (*top*) and our coupled algorithm (*bottom*). Our algorithm generally explores a smaller portion of the HFTS than the decoupled algorithm. In the *top* hierarchy the colored node is the only node associated with a feasible grasp. In the bottom the colors and numbers indicate the rating of the respective nodes. In contrast to the decoupled algorithm our algorithm only explored 6 leaves, of which 1 was a feasible grasp that was successfully connected to the forward tree.

$p_{max}(0.9)$	$p_{min}(0.01)$	$w_f(0.5)$	$w_c(4.0)$	$\kappa(4)$	$\omega_{1:6}(1)$
$p_{goalTree}(0.7)$	$D_m(0.4m)$	$N_a(20)$	$\alpha(10^{-6})$	$p_w(1.2)$	$\omega_{7:13}(0.25)$

Fig. 6: The chosen values for the different parameters.

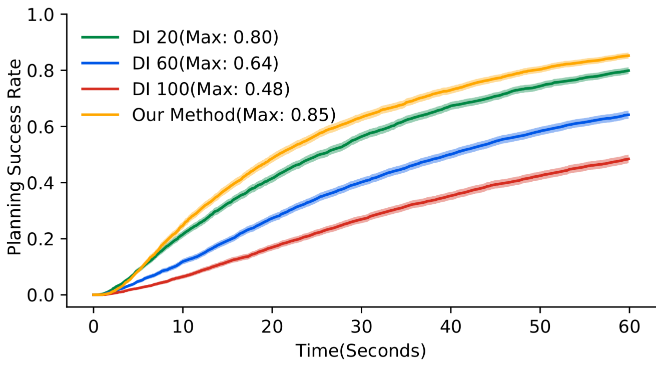

Fig. 7: The planning success rate as a function of planning time, which can be interpreted as a chance of planning success. The shaded areas show the Wilson confidence interval. DI: Decoupled integration with 20, 60, 100 iterations respectively.

dynamically adjusts the number of iterations i in the function SAMPLENEWCHILD in an interval $[i_{min}, i_{max}]$. For this interval we choose $i_{min} = 20$ and $i_{max} = 100$. In contrast to this, the decoupled algorithm performs a constant number of iterations per level. Therefore, for comparison we evaluate the decoupled algorithm for $i = 20, 60, 100$ iterations. In total we ran 4200 tests for each of these four algorithm configurations.

A. Planning Success Rate

For each test we record the runtime needed to successfully compute a solution or whether no solution is found within the time limit. As a result, we compute the ratio of successful tests as a function of planning time, which is shown in Fig. 7. As the runtime increases, more planning instances terminate successfully. The success rate of our algorithm dominates the success rate of all configurations of the decoupled algorithm,

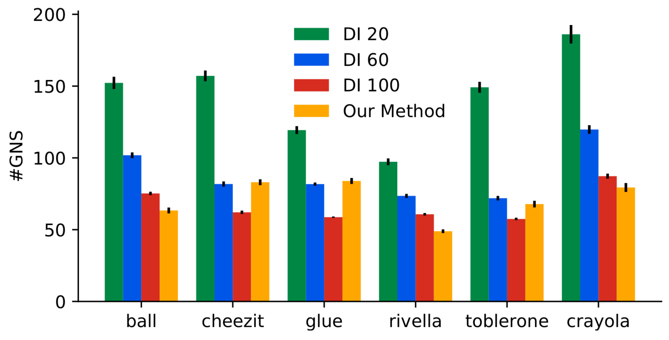

Fig. 8: Absolute number of HFTS nodes explored (GNS). DI: Decoupled integration with 20, 60, 100 iterations respectively. The black bars show the standard error.

achieving a maximum success rate of 0.85 at a total planning time of $60s$. In contrast, the decoupled algorithm achieves at most a final success rate of 0.8 for $i = 20$ iterations. For $i = 60$ and $i = 100$ the success rates are significantly worse. As a consequence the average runtime of our algorithm is $26s$, whereas the average runtime of the decoupled method is $30s, 38s, 45s$ for $i = 20, 60, 100$ respectively. Note that the decoupled method performs worse, the more expensive the grasp planning process is.

B. HFTS Exploration & Goal Planning Success

A low number of iterations allows the decoupled algorithm to explore more of the HFTS search space within the limited time budget. As there are fewer iterations on each level, the algorithm proceeds quicker to the bottom of the hierarchy. Consequently, we observe that the average number of HFTS nodes explored is the highest for the decoupled algorithm with $i = 20$ as shown in Fig. 8. The larger amount of explored HFTS nodes increases the chance of finding a grasp that is reachable and connectable by the motion planner.

In comparison, our method requires generally less exploration of the HFTS to find a reachable high quality grasp. Not only is the average number of explored HFTS nodes smaller, but also the chance of the goal sampler to compute a reachable grasp is significantly higher than for the decoupled method. This can be seen in Fig. 9, which shows the ratio between the number of times the goal sampler finds a valid goal configuration and the number of times the motion planning algorithm calls the goal sampler. This ratio can be interpreted as the chance of the respective goal sampling method to find a reachable solution. As the results show, this chance is the highest for our method for all objects. This indicates that our rating function successfully guides the goal sampling algorithm towards HFTS regions that are more likely to be reachable.

Interestingly, while all computed grasps are stable with respect to Canny-Ferrari's grasp quality, we do not observe any significant difference in grasp quality between all the methods with any number of iterations. This indicates that there are only few stable grasps reachable in our test environments, which achieve similar qualities.

V. DISCUSSION & CONCLUSION

In this work, we have presented an algorithm framework that is capable of simultaneous motion and grasp planning.

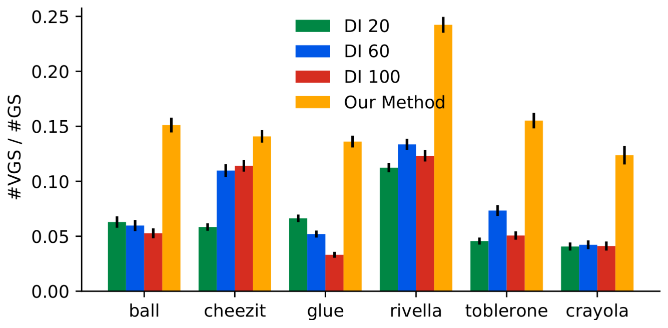

Fig. 9: The ratio between number of valid goal samples (VGS) and number of goal sample calls (GS). DI: Decoupled integration with $20, 60, 100$ iterations respectively. The black bars show the standard error.

The motion planning is performed on all DoFs of the robot, relieving us from defining any type of approach primitives. The contact based grasp planning allows us to purposefully search for stable fingertip grasps. In contrast to a decoupled integration, we showed that we can utilize the hierarchical nature of the grasp search to efficiently guide the algorithm towards regions of the HFTS that are likely to fulfill the reachability and connectability constraints. Our experimental evaluation showed that our method outperforms the decoupled integration in terms of average runtime, leading to a higher chance of planning success within a limited time frame. This improvement is particularly significant when the grasp computation is computationally expensive. Therefore, in future work we plan on evaluating our method on different hands with more fingers.

The rating function $t(n)$ we use for guiding the goal sampling is based on the proximity to known configurations in $\mathcal{C}_{free}$ or $\mathcal{T}_f \subset \mathcal{C}_{free}$. Hence, the degree to which the rating is informative strongly depends on the current state of exploration of $\mathcal{C}_{free}$. This makes the rating function in the beginning of the search not very informative. In future work, we plan to investigate alternative rating functions. In particular, we plan to investigate the usage of trajectory optimization methods [22]–[24] as relaxation for both constraints. These in combination with Bayesian optimization techniques [25] could be a promising alternative to our approach.

While the presented work focuses on integrating fingertip grasp with motion planning, we believe we can apply our framework to different applications. The framework makes few grasping related assumptions and requires only a hierarchical goal search space. One such potential application is placement planning, where a robot is tasked with placing an object safely in a cluttered environment. Furthermore, the framework could easily be extended to different varieties of bidirectional sampling-based motion planners such as [26] or applied to different grasp objective functions that, for instance, respect task constraints.

VI. Acknowledgments

This work was supported by the European Union framework program H2020-645403 RobDREAM.

References

[1] A. Bicchi and V. Kumar, "Robotic grasping and contact: A review," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000.

[2] M. Roa and R. Suárez, "Grasp quality measures: review and performance," *Auton. Robots*, vol. 38, no. 1, 2015.

[3] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis – a survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, 2014.

[4] J. Rosell, R. Suárez, and A. Pérez, "Path planning for grasping operations using an adaptive pca-based sampling method," *Auton. Robots*, 2013.

[5] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner, "Grasp planning in complex scenes," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2007.

[6] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Simultaneous grasp and motion planning: Humanoid robot armar-iii," *IEEE Robotics Automation Magazine*, 2012.

[7] J. Fontanals, B. A. Dang-Vu, O. Porges, J. Rosell, and M. A. Roa, "Integrated grasp and motion planning using independent contact regions," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, Nov 2014.

[8] K. Hang, J. A. Stork, and D. Kragic, "Hierarchical fingertip space for multi-fingered precision grasping," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2014.

[9] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[10] K. Hang, J. A. Stork, F. T. Pokorny, and D. Kragic, "Combinatorial optimization for hierarchical contact-level grasping," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014.

[11] C. Ferrari and J. Canny, "Planning optimal grasps," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1992.

[12] F. T. Pokorny and D. Kragic, "Classical grasp quality evaluation: New theory and algorithms," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2013.

[13] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: representing robot capabilities," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2007.

[14] ——, "Online generation of reachable grasps for dexterous manipulation using a representation of the reachable workspace," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009.

[15] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann, "Humanoid motion planning for dual-arm manipulation and re-grasping tasks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2009.

[16] D. Berenson, S. Srinivasa, D. Ferguson , and J. Kuffner, "Manipulation planning on constraint manifolds," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2009.

[17] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[18] J. A. Haustein, J. King, S. S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2015.

[19] J. E. King, J. A. Haustein, S. S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2015.

[20] P. C. Chen and Y. K. Hwang, "Sandros: a dynamic graph search algorithm for motion planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 3, Jun 1998.

[21] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, 2010.

[22] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011.

[23] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, 2013.

[24] M. Mukadam, X. Yan, and B. Boots, "Gaussian process motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2016.

[25] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *CoRR*, vol. abs/1012.2599, 2010.

[26] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. MIT-CSAIL-TR-2013-021, August 2013.