Interleaving Planning and Control for Efficient Haptically-guided Reaching in Unknown Environments

Daehyung Park, Ariel Kapusta, Jeffrey Hawke, and Charles C. Kemp

***Abstract*— We present a new method for reaching in an initially unknown environment with only haptic sensing. In this paper, we propose a haptically-guided interleaving planning and control (HIPC) method with a haptic mapping framework. HIPC runs two planning methods, interleaving a task-space and a joint-space planner, to provide fast reaching performance. It continually replans a valid trajectory, alternating between planners and quickly reflecting collected tactile information from an unknown environment. One key idea is that tactile sensing can be used to directly map an immediate cause of interference when reaching. The mapping framework efficiently assigns raw tactile information from whole-arm tactile sensors into a 3D voxel-based collision map. Our method uses a previously published contact-regulating controller based on model predictive control (MPC). In our evaluation with a physics simulation of a humanoid robot, interleaving was superior at reaching in the 9 types of environments we used.**

I. Introduction

Consider a visually hard-to-observe environment, such as a room in total darkness. If the environment is not empty space, it may be hard to reach a target location. Your reach may make contact over the surface of your arm as you grope towards your target. As you encounter objects, you piece together environmental information that allows you to find a path to the goal. You may also begin with a simple greedy reaching strategy. If you encounter obstacles and are unable to reach your goal, you are likely to increase attention your attention to the task.

Although many studies have investigated robotic reaching in readily observable or known environments, there are few studies of situations that only permit haptic sensing. Moving a robot arm in cluttered, unknown, and unmodeled workspaces can be difficult. What are the roots of the problem? Firstly, there is no fast, complete sensing method to fully map unknown objects. Some works have used the senses of sight and hearing to map environments (See section II). However, vision or acoustic sensors are affected by factors such as external sources, occlusions, and mounting locations. Second, when reaching into unknown environments, there can be unpredictable collisions. These collisions lower the tracking ability of a controller and may result in high contact forces that cause damage to the robot or environment. Third, geometric planners require knowledge about obstacles in the environment in order to avoid them.

To solve this reaching problem, we present a few key ideas. First, we cover the robot arm entirely with tactile sensors that measure forces between the robot and the environment. As the arm encounters obstacles we use a haptic-cost map generator to create a partially observed obstacle map, called a haptic-cost map. Second, we propose a haptically-guided interleaving planning and control method (HIPC) (see Fig. 2). HIPC contains two sequentially executed planners: a task-space and a joint-space planner. Each is an interleaving planning and execution (IPE) system that interleaves running a planner and a controller that limits contact forces [1], [2]. The planner continually plans a trajectory based on the current haptic-cost map. The controller follows the planned trajectory until given a new trajectory.

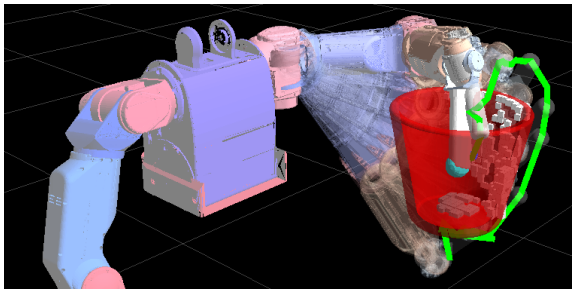


Fig. 1: *A screenshot of our simulated robot (DARCI) haptically reaching into an unknown basket. The red colored transparent object is a fixed, rigid, and unknown upright basket. A blue sphere shows the goal location. The left arm explores from the bottom of the basket and finally reaches the goal location using a haptic-cost map shown as white voxels.*

Our approach is inspired in part by the human experience of performing a manipulation task without paying close attention, noticing that something has gone wrong, and then carefully and deliberately attempting to perform the task. In our system, the task-space planner continually replans a Cartesian space trajectory for the end effector of the robot arm and follows the trajectory, often snaking around objects as it regulates contact forces. This task-space approach is computationally low-cost, can take advantage of joint redundancy, and has performed well in high clutter [2], [3]. However, it can become trapped in local minima due to its greedy policy. The joint-space approach complements this task-space approach by escaping local minima and being probabilistically complete. It gives a global solution with respect to the current haptic-cost map at a higher computational cost. Then, a joint-space contact-regulating controller, which is an updated version of our dynamic model predictive control (MPC) system from [2], tracks this joint-

Healthcare Robotics Lab, Georgia Institute of Technology, Atlanta, GA. D. Park is the corresponding author {deric.park@gatech.edu}.

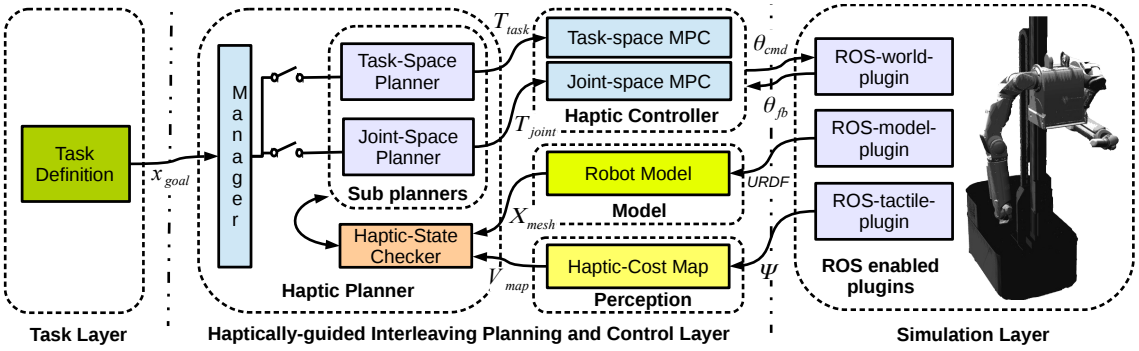


Fig. 2: *Overview of our haptically-guided interleaving planning and control (HIPC) system. HIPC switches between task-space planning and control and joint-space planning and control to reach a goal location x_{goal}. The planners use a haptic-state checker based on a haptic-cost map. Ψ is tactile feedback. θ_{cmd} and θ_{fb} are robot arm input and output.*

space trajectory.

We evaluated our approach in a variety of static environments by physically simulating a humanoid robot in Gazebo (http://gazebosim.org). Our HIPC method outperformed the independent use of the task-space approach and the joint-space approach. As our results show, by interleaving these two approaches, our system uses computational resources more economically than if it were to only use joint-space planning, and more frequently finds solutions than if it were to only use task-space planning. Notably, our system favors exploitation over exploration. At all times, it is doing its best to reach the goal rather than producing a full map of the environment. We have also proved the completeness of an idealized version of HIPC.

II. Related Works

The concept of interleaved planning and execution has a long history, as noted in Ambros-Ingerson and Steel's paper [1]. For example, Nourbakhsh and Reza classified interleaving strategies into two categories; *subgoaling* and *simplification* [4]. In this paper, we use the *simplification* strategy that returns a complete plan in a limited period with an approximated problem. We use our map of the partially observed environment for the approximation. Recently, Park et al. showed real-time replanning and execution using a graphics processing unit (GPU) [5]. Kaelbling and Lozano-Perez have investigated hierarchical replanning and execution with geometric reasoning [6]. Our work is also related to online replanning. For example, Koenig and Likhachev presented Dynamic A* Lite that is an incremental heuristic search algorithm [7]. This algorithm speeds up their replanning searches using the experience of previous problems, which relates to our use of task-space planning.

Our approach is strongly related to sensor-guided real-time planning methods. Rusu et al. proposed a 3D perception-based replanning architecture using a voxel-based collision map constructed from a laser range-finder [8]. For replanning, they presented an offline replanning method using a sampling-based planner. In contrast, we combine an online replanning method with a search-based planner to compensate for the limited sensing coverage of the tactile sensor and we integrate it with a haptic-based reactive controller. Um et al. proposed a simultaneous planning and mapping framework using a non-contact 3D depth sensor [9]. The method is similar in that two mapping and planning processes are running in parallel, and it uses a configuration-space planner to achieve probabilistic completeness. In contrast, we only use haptic sensing and sequentially use task-space and configuration-space planners. Research, such as [10], has looked at the problem of registering a pre-existing 3D model to the world based on tactile sensing. In contrast, our method does not use a pre-existing model of the world.

We have previously demonstrated the feasibility of whole-arm tactile sensing, planning, and control on a real robot [11]. In the current paper, we present HIPC, which formalizes aspects of this previous proof-of-concept system. In addition, we present carefully controlled tests based on thousands of trials in simulation, as well as a proof of completeness.

III. A Haptically-guided Planner

A. Notation and Definition

Let $\mathcal{X}$ denote the task space that is an open subset of $\mathbb{R}^3$. To represent the haptic-cost map, we project $\mathcal{X}$ into a discretized three dimensional grid (called a "voxel grid"), denoted by $\mathcal{V}$, where we define the map as $\mathcal{V}_{map} \subseteq \mathcal{V}$. We also use joint space as configuration space (C-space), denoted by C. C is an open subset of $\mathbb{R}^n$, where n is the number of degrees of freedom (DoF) of the robot arm. A state in $\mathcal{X}$ and C is valid only when it is collision-free and reachable by the arm. The computed trajectories in task-space and joint-space are denoted by $\mathcal{T}_{task} \subset \mathcal{X}$ and $\mathcal{T}_{joint} \subset C$, respectively.

B. Overview of Planning Architecture

The haptic planner consists of two sub planners; a task-space and a joint-space planner. Both sub planners and their associated controllers use the architecture depicted in Fig. 2. For example, given a robotic system with tactile skin, a task definition module specifies a goal location $x_{goal} \in \mathcal{X}$, then a manager runs the sub planners in turn, where the planners return $\mathcal{T}_{task}$ or $\mathcal{T}_{joint}$ to reach x_{goal}. Each planner sends its trajectory to a corresponding controller that is responsible for following the trajectory while keeping contact forces low.

The robot with simulation plugins provides its current state and tactile feedback to the controller and perception modules.

The state contains the location of the end effector $x \in \mathcal{X}$ and joint configuration $q \in C$. The tactile feedback is a set of locations and magnitudes of contact forces. The perception module constructs $\mathcal{V}_{map}$ from the tactile sensing skin. The robot model module uses the robot's kinematic parameters to compute various forms of kinematic information, such as forward kinematics and the locations of arm mesh vertices $\mathcal{X}_{mesh}$ for the validity check. A haptic-state checker inspects the validity of each state in a trajectory using the robot model and the haptic-cost map modules. If any state is invalid, the planner computes a new trajectory.

C. Planning in Detail

Algorithm 1 shows how the manager alternates the sub planners and interleaves those with MPC. In the first half of the algorithm, **TaskSpacePlan()** generates $\mathcal{T}_{task}$. **ExecuteMPC()** asynchronously runs task-space MPC to track the computed $\mathcal{T}_{task}$ in parallel to Algorithm 1 until it is called again with a new plan. The repetition of task-space planning and execution results in rapid adaptation in response to tactile sensing. After the haptic-map updates, the task-space planner quickly returns a task-space trajectory to the goal. Task-space MPC runs in a parallel thread at 20 Hz and takes advantage of redundant degrees of freedom to snake around obstacles while following the task-space trajectory, as described in [2]. This task-space subsystem stops when the end effector reaches a goal region $\mathcal{X}_{goal} \subset \mathcal{X}$, t_{max} time elapses, or the arm stops moving. The system decides that the arm is stuck if the magnitude of the angular difference between current and past moving averaged joint angle vectors is smaller than a threshold c.

If the task-space subsystem fails, **JointSpacePlan()** generates $\mathcal{T}_{joint}$. **ExecuteMPC()** asynchronously runs joint-space MPC to follow $\mathcal{T}_{joint}$ in parallel to Algorithm 1. The algorithm then monitors the robot's progress, taking appropriate action in the event of success or failure. Due to the higher computational requirements associated with generating a joint-space plan and the limited ability of joint-space MPC to move around unforeseen obstacles, the joint-space subsystem tends to be less adaptive and reactive than the task-space subsystem. However, the joint-space planner generates a collision-free trajectory that takes into account the volume of the arm, which can provide solutions in situations that cause the task-space subsystem to fail.

By combining these two subsystems, HIPC leverages the computational efficiency of the task-space planner, the reaching speed of task-space MPC, and the probabilistic completeness of the joint-space planner. We now describe the planners in detail.

1) Task-space Planner: **TaskSpacePlan()** is an adapted version of the A^* search algorithm for repetitive planning. Since A^* uses a graph data structure, we define the geometric structure of $\mathcal{X}$ as a graph, $\mathcal{G} = (V, E)$, consisting of a set V of vertices, also called nodes, and a set E of edges. Using a cell decomposition approach, we represent V and E as the sets of voxels and connections between adjacent voxels, respectively [12]. The center location of each voxel $v \in V$ is denoted by $x_v \in \mathcal{X}$. Algorithm 2 shows the detailed flow, where an evaluated cost function is represented as $f(v) = g(v) + h(v)$, where $g(v)$ is a path-cost function and $h(v)$ is an admissible heuristics using Euclidean distance. Finally, **TracePath()** returns a minimum-cost path from the graph.

Algorithm 1: Interleaving Planning and Control

Input : x_{goal}

$q, x \leftarrow$ `GetCurrentState ()` ;
while *true* **do**
  $t \leftarrow 0$;
  repeat
    $\mathcal{T}_{task} \leftarrow$ `TaskSpacePlan(`q, x_{goal}`)` ;
    `ExecuteMPC(`$\mathcal{T}_{task}$`)` ;
    $q, x \leftarrow$ `GetCurrentState ()` ;
    $\hat{q}_t \leftarrow$ `GetMovingAvg (`q`)` ;
    if $x \in \mathcal{X}_{goal}$ **then**
      return success ;
  until !($t > t_{max}$ **or** $\|\hat{q}_t - \hat{q}_{t-k\Delta t}\| < c$**)**;
  $\mathcal{T}_{joint} \leftarrow$ `JointSpacePlan(`q, x_{goal}`)` ;
  `ExecuteMPC(`$\mathcal{T}_{joint}$`)` ;
  $t \leftarrow 0$;
  repeat
    $q, x \leftarrow$ `GetCurrentState ()` ;
    $\hat{q}_t \leftarrow$ `GetMovingAvg (`q`)` ;
    if $x \in \mathcal{X}_{goal}$ **then**
      return success ;
  until !($t > t_{max}$ **or** $\|\hat{q}_t - \hat{q}_{t-k\Delta t}\| < c$**)**;

Our system frequently replans to avoid newly found contact locations. We reduce expansion time by storing and ignoring a list of invalid nodes V_{inv}, which is appropriate for static environments. Also for efficiency, we do not take into account the geometric constraints of the arm. Instead, the planner finds a trajectory for an approximated representative volume of the end effector, a sphere of radius r, located at the tip of the end effector (see Fig. 3). **IsStateValid()** is a state checker that inspects the validity of a specified state discussed in Section III-D.

2) Joint-space Planner: **JointSpacePlan()** is a sampling-based planner, which is different from the task-space planner in that it takes into account the entire volume of the arm. Algorithm 3 shows the sequence to get $\mathcal{T}_{joint}$, where the sequence is divided into two parts: a goal configuration selection step and a trajectory planning step.

The goal configuration selection step finds a least-cost and valid target configuration q_{goal} from a set of randomly sampled configurations, $\mathbf{q}_{rand}$. The set contains multiple IK solutions from a set of random end-effector poses $(x_{goal}, \boldsymbol{\eta}_{rand})$, where $\boldsymbol{\eta}_{rand}$ is a set of uniformly distributed quaternions

Algorithm 2: TaskSpacePlan(q_{init}, x_{goal})

Static : V_{inv} *is invalid node list and initially* $\emptyset$
Local : $V_o \leftarrow \emptyset$ *open node list*
Local : $V_c \leftarrow \emptyset$ *closed node list*
Set v_{init} as a voxel of $x_{init} \leftarrow$ `FwdKin(`q_{init}`)` and add to V_o ;
Set v_{goal} as a voxel of x_{goal} ;
repeat
  Pick $v_{best} \in V_o$ such that $f(v_{best}) \leq f(v)$, $\forall v \in V_o \setminus \{v_{best}\}$;
  Remove v_{best} from V_o and add to V_c ;
  if $v_{best} = v_{goal}$ **then**
    return $\mathcal{T}_{task} \leftarrow$ `TracePath(`v_{init},v_{goal}`)` ;
  Expand $\forall v \in neighbor(v_{best})$ and $v \notin V_c, V_{inv}$;
  if !`IsStateValid(`v`)` **then**
    add v to V_{inv} ;
  else if $v \notin V_o$ **then**
    add v to V_o ;
  else if $g(v_{best})$+`Distance(`v_{best},v`)` $< g(v)$; **then**
    update v's path cost and backpointer ;
until V_o *is empty*;

using K. Shoemake's algorithm [13]:

$$\begin{aligned} \boldsymbol{\eta}_{rand} = \{(x,y,z,w) | x &= \sqrt{1-u_1} \cdot sin(2\pi u_2) \\ y &= \sqrt{1-u_1} \cdot cos(2\pi u_2) \\ z &= \sqrt{u_1} \cdot sin(2\pi u_3) \\ w &= \sqrt{u_1} \cdot cos(2\pi u_3)\}, \end{aligned} \tag{1}$$

where u_1, u_2, and u_3 are independent uniform random values between 0.0 and 1.0. For each end-effector pose, we find a single IK solution based on the robot's current configuration and using velocity-based inverse kinematics via the Kinematics and Dynamics Library (http://www.orocos.org). Ideally, we would instead sample from all feasible configurations. We exclude invalid states using **IsStateValid()**. The closest configuration to initial configuration q_{init} is selected by weighted Euclidean distance:

$$c(q_1, q_2) = (q_1 - q_2)^T W (q_1 - q_2), \tag{2}$$

where W is a positive semi-definite weight matrix, and q_1 and q_2 are joint-space vectors.

In the trajectory planning step, we use a randomized-sampling motion planner, RRT-Connect [14], although our approach could use other planners. The planner computes a series of states, sums the collision costs along the trajectory using the haptic-state checker (see section III-D), and returns the trajectory if the cost is not over some threshold ζ_c. If the planner fails to find a trajectory within 30s, it repeats the planning step with another goal configuration until it finds a valid trajectory or attempts 10 different configurations. In this paper, we use the first successfully generated trajectory to decrease execution time.

Algorithm 3: JointSpacePlan(q_{init},x_{goal})

repeat
  $\mathbf{q}_{rand} \leftarrow IK(x_{goal}, \boldsymbol{\eta}_{rand})$;
  Pick $q_{goal} \in \mathbf{q}_{rand}$ such that $c(q_{goal}, q_{init}) < c(q, q_{init})$, $\forall q \in \mathbf{q}_{rand} \setminus \{q_{goal}\}$ and `IsStateValid(`q_{goal}`)` ;
  $\mathcal{T}_{joint} \leftarrow$ `RRT-Connect(`q_{init},q_{goal}`)` ;
  if $\mathcal{T}_{joint}$ *exists* **then**
    Reduce vertices and interpolate $\mathcal{T}_{joint}$;
    if `IsTrajValid(`$\mathcal{T}_{joint}$`)` **then**
      Return $\mathcal{T}_{joint}$;
until $\mathcal{T}_{joint}$ *is not empty*;

D. Haptic-state Checker

Our system allows contact between the robot and obstacles, which, in practice, complicates decisions about the validity of a state or trajectory. The haptic-state checker consists of the functions **IsStateValid()** and **IsTrajValid()**, which use collision costs and the robot's workspace to decide on the validity of a state or trajectory. The collision cost is the estimated degree of penetration of the robot volume into obstacle volume on $\mathcal{V}_{map}$. Each voxel in $\mathcal{V}_{map}$ contains a collision-cost representing the hardness of passing through the voxel. If the accumulated cost ζ along the arm links is non-zero, the robot may be in an invalid state.

A variety of factors could result in a state that is achievable by the robot being labeled as invalid, including sensor uncertainty, object deformation, and simulation error. To mitigate these issues, we define *soft-collision* as a collision of two bodies with a small amount of penetration. When ζ is smaller than a threshold ζ_c, we judge the state valid. To compute the penetration, we use a VERTEX-VERTEX collision detection algorithm [15] that declares a collision if two vertices are within a certain range. We sample $\mathcal{X}_{mesh}$ from the surface of the robot collision mesh. Then, we find cost voxels $v_x \in \mathcal{V}$, which contain $x \in \mathcal{X}_{mesh}$. We are able to compute the degree of collision ζ by summing the values of the voxels (see Fig 3).

In addition, we check if $\mathcal{X}_{mesh}$ is in the workspace, $\mathcal{X}_{workspace} \subseteq \mathcal{X}$. If at least one vertex is outside the workspace, the target state is invalid. Algorithm 4 is the pseudo code of **IsStateValid()** function, where **GetCollisionCost(**v_x**)** returns the collision cost of voxel v_x in $\mathcal{V}_{map}$. The algorithm calculates ζ, the sum of collision cost of all mesh voxels. If ζ exceeds ζ_c the state is invalid. **IsTrajValid()** is an extended version of **IsStateValid()** that computes the sums the values of ζ from all states in a sampled trajectory. If the sum is greater than ζ_c, the trajectory is invalid.

Algorithm 4: IsStateValid(state s)

$s_{init} \leftarrow$ initial state such that x_{init} **or** q_{init} ;
if $s_{init} == s$ **then**
  return true ;
else
  $\mathcal{V}_{map} \leftarrow$ `GetHapticMap()` ;
  $\mathcal{X}_{mesh} \leftarrow$ `GetRobotMesh(`s`)` ;
  $\zeta \leftarrow 0$;
  for $x \in \mathcal{X}_{mesh}$ **do**
    if $x \notin \mathcal{X}_{workspace}$ **then**
      return false ;
    Pick $v_x \in \mathcal{V}$ that contains x in Cartesian space ;
    $\zeta = \zeta +$ `GetCollisionCost(`v_x`)` ;
    if $\zeta > \zeta_c$ **then**
      return false ;
  return true ;

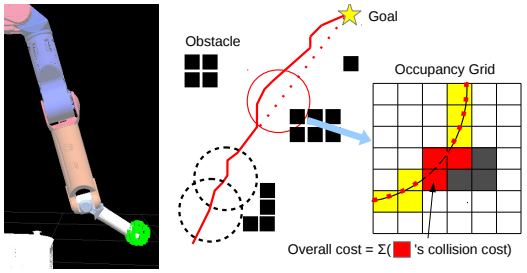


Fig. 3: ***Left:*** *Sampled vertices of a sphere for task-space planning.* ***Right:*** *Example of a collision-cost computation on a 2D grid. Grey cells represent detected objects with non-zero cost. Yellow cells indicate voxels that contain the sampled vertices. Red cells show the overlapped area of the two types of cells.*

IV. A Haptic Environmental Mapper

A. Contact forces to tactile feedback

Before we explain map construction, we discuss tactile feedback. The contact force, denoted by f, consists of a normal force f_n and a tangential force f_t. A contact force is generated when the surfaces of bodies come in contact. In the dynamic simulator, Gazebo (http://gazebosim.org), the surface of an object in simulation is a large set of triangular meshes, called *TriMeshes*. Inter-penetration of two *TriMeshes* creates f proportional to the degree of penetration. Here, we define raw contact data as a tuple, (p, f, f_n), where p is a contacted location. We are able to utilize this data for haptic control and mapping, but its direct utilization is computationally expensive. We simplify the raw data into a form we call *tactile feedback*.

Fig. 4 (Left) illustrates this simplification. We first generate k clusters from the data based on the contact locations, p, using k-means clustering via Lloyd's algorithm in the KMlocal library (http://www.cs.umd.edu/∼mount/). Let $\hat{n}_i$

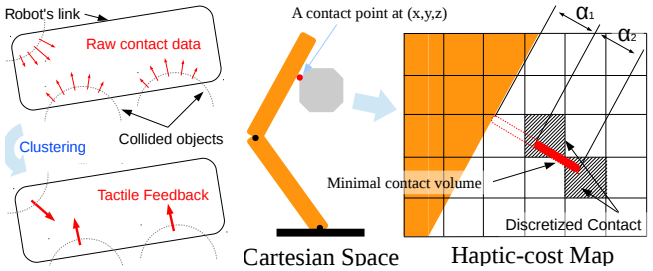


Fig. 4: ***Left:*** *Clustering of raw contact data into tactile feedback.* ***Right:*** *Haptic mapping on a 2D grid. A red segment shows the minimal volume associated with any contact, which is placed away from the arm to avoid overlap between the arm and the map.*

denote a unit vector of the sum of f_n in cluster i. Then we compute a representative tuple for each cluster,

$$(p_i, f_i, f_{n,i}) = (\sum_{j=1}^{m} p_j/m, \sum_{j=1}^{m} f_j, (f_i \cdot \hat{n}_i)\hat{n}_i), \qquad (3)$$

where $\forall i \in \{1, ..., k\}$ and m is the number of raw samples in cluster i. To distinguish different types of objects (e.g. movable, compliant, fixed objects) we append the collision cost γ to the tuple: (p, f, f_n, γ). A higher cost represents a higher stiffness to push or pass a contact. In this paper, we only use fixed objects with a uniform cost. Tactile sensing could be used to assign γ values in real time [16], [11].

B. Tactile feeback to a haptic-cost map

Our mapping approach assigns the collision-costs from tactile feedback to the voxels in $\mathcal{V}_{map}$. Rather than use a single voxel, we define a *minimum contact volume* $\mathcal{B}$, which represents the minimal volume associated with any contact detected by the arm (see Fig. 4 (Right)). In this paper, $\mathcal{B}$ is a line segment between $p + \alpha_1\hat{n}$ and $p + (\alpha_1 + \alpha_2)\hat{n}$, where $\hat{n}$ is a unit vector of f_n. α_1 and α_2 are a penetration prevention offset.

V. Haptically-Guided Control

We use the multi-step model predictive control (MPC) system described in [2] and [11] running at 20 Hz on our simulated robot. The controllers move the robot arm towards a goal end effector position or goal joint-space configuration, while keeping predicted contact and impact forces low. It has a control horizon of 3 and a prediction horizon of 4, which gives the controller 4 time steps (each step is 0.05 seconds) of control (including the current time) and predicts the arm movement for 4 additional time steps over which it aims to minimize its cost function. The controller only executes the current time-step before running the optimizer again.

VI. Proof of Completeness

We now prove completeness for an idealized version of HIPC. Specifically, we prove that an idealized version of HIPC is guaranteed to eventually find a path from an initial configuration, q_{init}, to a final configuration, q_{goal}, if a tractable path exists in a discretized version of a static

environment. We also prove that HIPC will report failure if no solution exists in the discretized environment.

For a given environment, we define M_{all} to be a set that represents a 3D occupancy grid for the environment with discretization at resolution d. Each element of the set represents a voxel that is occupied if an object in the environment intersects any part of the voxel. $Q_{all} = S_c(M_{all}, q_{init}, q_{goal})$, where S_c is a joint-space planner with proven completeness and Q_{all} is a joint-space trajectory that when followed takes the robot from q_{init} to q_{goal} without intersecting any occupied voxel in M_{all}. We assume that the controllers we use result in the robot perfectly following any trajectory. We also assume the robot perfectly detects when it is about to enter an occupied voxel of M_{all} and stops the robot just before intersection occurs. Thus, the robot never intersects an occupied voxel of M_{all} [1].

$P(M_{all})$ is the set of all subsets of M_{all} (i.e. the power set of M_{all}). Q_{all} is a solution trajectory from q_{init} to q_{goal} for each element of $P(M_{all})$. At any time, the robot can follow any sequence of trajectories it has taken in reverse in order to return to q_{init}. Thus, S_c is guaranteed to return a solution trajectory for any element of $P(M_{all})$ from the robot's current configuration q_i to q_{goal}, since the robot can always return to q_{init} and then follow Q_{all}.

At each iteration of HIPC, the robot uses its current map of the world M_i to plan a trajectory $Q_i = S_c(M_i, q_i, q_{goal})$. It starts with $Q_0 = S_c(M_0, q_0, q_{goal})$, where $q_0 = q_{init}$ and $M_0 = \emptyset$, which has cardinality of 0, $|M_0| = 0$. The robot then attempts to follow Q_i. It either reaches q_{goal} or halts. If it halts, it adds the occupied voxels that caused it to halt to its map, so $|M_{i+1}| > |M_i|$. Since M_{i+1} is an element of $P(M_{all})$, the cardinality of M_i strictly increases, and the maximum cardinality of any element of $P(M_{all})$ is $|M_{all}|$, HIPC is guaranteed to result in the robot reaching q_{goal}.

If no solution exists, S_c will report failure for some M_i, which can only occur if Q_{all} does not exist. Since our proof did not use the resolution, d, it holds for any resolution, although only some resolutions may have solutions.

After following joint-space plan Q_i, the robot may follow any finite-time joint-space trajectory without changing the result of this proof. The robot either reaches q_{goal} or adds new occupied voxels that resulted in halting to M_{i+1}. M_{i+1} must still be an element of $P(M_{all})$ and $|M_{i+1}| > |M_i|$. Since the robot can always follow the reverse of a finite-time trajectory to return to q_i, S_c is guaranteed to find a plan, $Q_{i+1} = S_c(M_{i+1}, q_{i+1}, q_{goal})$, if a solution exists. We can model our task-space planner with task-space control as a subsystem that results in the robot following a sequence of finite-time joint-space trajectories, where halting and mapping occur between each trajectory in the sequence. Thus, task-space planning and control do not change the completeness of the overall algorithm.

[1]This avoids problems with finite-precision environment representations, which can otherwise result in a trapped robot. Additive small-scale space-filling robot motions, short-range sensing without rigid contact, high-resolution maps, and backtracking could all help a real robot meet this requirement.

For our implementation, we use a probabilistically complete joint-space planner that will eventually return a solution if one exists, but will not report when a solution does not exist. With minor modification of our proof, one can show that our idealized version of HIPC would be probabilistically complete.

VII. Experimental Setup

We use a physical simulation of our robot DARCI, which is a Meka M1 Mobile Manipulator. The simulated robot includes an omnidirectional base with the two humanoid, 7 DoF MEKA M1 arms shown in Fig. 1. We only moved the left arm (7 DoF) in our experiments. Our system uses ROS (http://ros.org). To detect collisions, we use virtual contact sensors on the entire surface of the left arm links that return raw contact data. We developed three ROS-enabled-plugins: Gazebo-world-plugin, Gazebo-model-plugin, and Gazebo-tactile-plugin[2]. The Gazebo-world-plugin provides a synchronization interface with ROS, the Gazebo-model-plugin provides an interface for the *Joint Trajectory Action Controller* of ROS at 1 KHz, and the Gazebo-tactile-plugin generates tactile feedback messages from the raw contact data at 30 Hz. We ran the simulation on multiple machines, each with an i7 3.4 GHz CPU and 8 GB of memory.

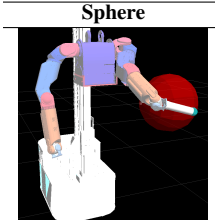
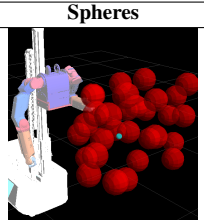
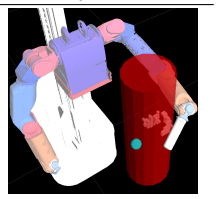
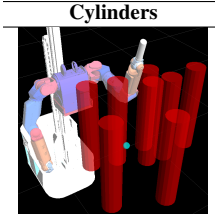
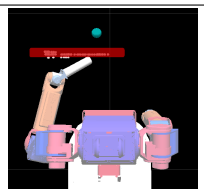
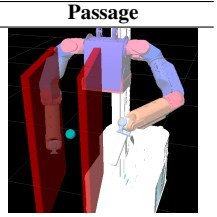
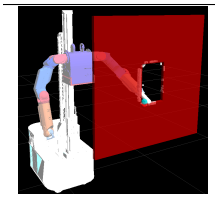
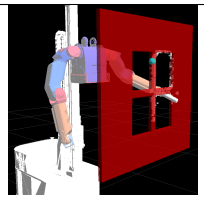
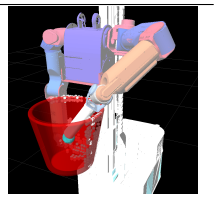
A. Test Environments

To evaluate our method, we used nine types of parametric clutter (see Table I). For each type of clutter, we created 1000 specific test environments with parameters randomly sampled from uniform distributions. Only the geometry varied, since we used the same physics simulation parameters for all objects. The objects were rigid and fixed. Descriptions of some of the dimensions and random variation for these environments follow:

- **Sphere**: A sphere with 0.12 m radius randomly placed in a 0.1 $\times$ 0.4 $\times$ 0.3 m (length $\times$ width $\times$ height) volume. Goal is located 0.03 m behind the sphere.
- **Spheres**: 30 spheres with 0.08 m radius randomly placed in a 0.5 $\times$ 0.9 $\times$ 0.6 m volume. Goal is randomly located in a 0.3 $\times$ 0.55 $\times$ 0.35 m volume in front of DARCI.
- **Cylinder**: Upright cylinder of radius 0.06 m and height 0.4 m randomly placed in a 0.15 $\times$ 0.5 m (length $\times$ width) area. Other conditions same as **Spheres**.
- **Cylinders**: Eight upright cylinders randomly placed in rectangular area, 0.5 $\times$ 0.9 m. Other conditions same as **Cylinder**.
- **Wall**: A wall, 0.03 $\times$ 0.2 $\times$ 0.6 m, randomly placed in the rectangular area, 0.2 $\times$ 0.6 m. Goal is located 0.1 m behind the wall at 0.1 m height.
- **Walls**: Two parallel walls, 0.03 $\times$ 0.15 $\times$ 0.6 m, with 0.3 m offset randomly rotated by $\pm 30^\circ$. Goal is located in between them.
- **Window**: A wall with a single window, 0.3 m width $\times$ 0.2 m height, randomly placed in a 0.1 $\times$ 0.4 $\times$ 0.3 m volume. Goal is located at least 0.05 m behind the window.

[2]We've released our Gazebo-tactile-plugin as open source code at *gt-ros-pkg*: https://github.com/gt-ros-pkg/gt-meka-sim.

TABLE I: Examples of cluttered environments. Obstacles are red. The goal is visualized as the small blue sphere.

Sphere	Spheres	Cylinder
		
Cylinders	**Wall**	**Passage**
		
Window	**Windows**	**Trash can**
		

- **Windows**: A wall with four windows, 0.25 $\times$ 0.2 m with 0.05 m offset, randomly placed in the same way as the **Window**.
- **Trash can**: A round basket with a goal inside it and a 0.1 m radius opening randomly placed in a 0.1 $\times$ 0.4 $\times$ 0.15 m volume.

B. Evaluation Scheme

We randomly selected an initial end effector location on a vertical, rectangular plane of 0.6 m width and 0.2 m height in front of the robot. Then, we chose the robot arm's initial configuration using a cost-metric function described in [17]. We ran, under the same conditions, three different types of planning methods: HIPC, task-space planning, and joint-space planning. We assigned success or failure labels according to whether the end effector reached within a 0.02 m radius spherical volume around the goal within 1,000 seconds.

To compare the performance of HIPC with a complete planning method in a known environment, we estimated the maximum success rate of bi-directional RRT using OpenRave (http://openrave.org). This planner started with complete knowledge of the test environment. If the planner could find a solution in that environment without collision in fewer than 7,000 iterations, we considered the clutter solvable. To take into account the ball-shaped goal area for the planner, we created a set of goal configurations. We chose 72 evenly distributed 3D locations on the surface of the ball and at each location, we selected 512 different quaternion vectors using the *sampleSO3* function in OpenRAVE. We then found a single IK solution for each of these end effector poses. While this is an underestimate for the maximum success rate, Figure 7 clearly illustrates that close to 100% of the tasks were solvable.

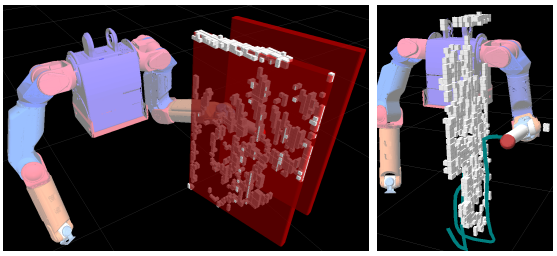


Fig. 5: *Screenshot of haptic mapping and planning in a* ***Walls*** *type test environment. The white voxels show the haptic-cost map. The green line shows the path of the end effector for the last seconds before the robot reached the goal.*

VIII. Results

As an example, Fig. 5 (Left) illustrates a reaching experiment in a **Walls** type test environment. The image depicts the placement of the robot and clutter, and its left arm's initial configuration in front of the clutter. The white voxels inside the two walls represent the detected contacts through the 565 seconds of HIPC execution. The green line in Fig. 5 (Right) shows the last planned task-space trajectory for the end effector along with the haptic-cost map used for planning, $\mathcal{V}_{map}$. These figures illustrate that the robot may only need to sparsely map the environment in order to reach the goal.

Fig. 6 shows the number of unique contacts that occurred over time in order to visualize the growth of the robot's map, which relates to the proof in Section VI. The red and blue shadings in the graph indicate the periods of task-space and joint-space control, respectively. The periods of task-space control are often longer than the periods of joint-space control. This is in part due to task-space control continuing to be active until the joint-space planner returns a plan. The upper graph represents a successful reach. Although the task-space planner by itself is not complete, HIPC benefits from its fast planning and reactive movements. For example, the haptic-cost map grows while the task-space subsystem is active. HIPC also benefits from the task-space controller's ability to navigate around and through contact well. The lower graph shows a failed trial wherein HIPC fell into a degenerate loop. The graph is from the robot arm failing to sufficiently track the planned joint-space trajectory and continually colliding with already-mapped obstacles. As a result HIPC could not collect new contact data and repeatedly generated similar plans. This illustrates how things can break down due to imperfect trajectory tracking.

Next, we compared the performance of HIPC to the two sub methods; a task-space and a joint-space planner. All three methods used the same initial configuration and goal locations in the same environment. Fig. 7 shows HIPC had a higher success rate than the other two methods in all nine types of environment. Over the total 9,000 trials (1000 trials per environment), HIPC attained a 93.1% success rate. In contrast, the task-space and joint-space planners resulted in

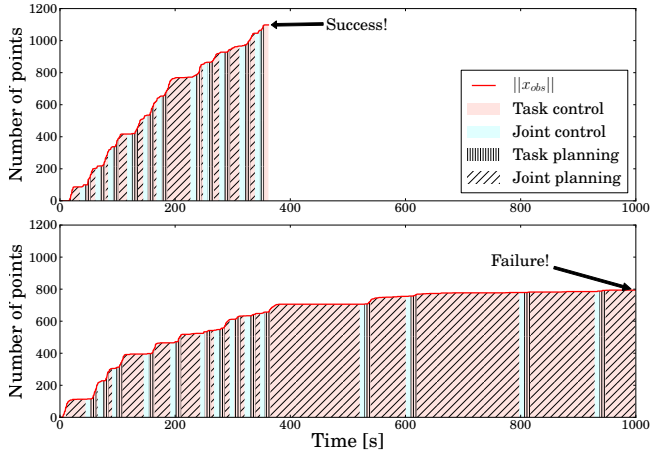


Fig. 6: *The number of unique contacts that occurred over time while reaching in two distinct* ***Walls*** *type environments.*

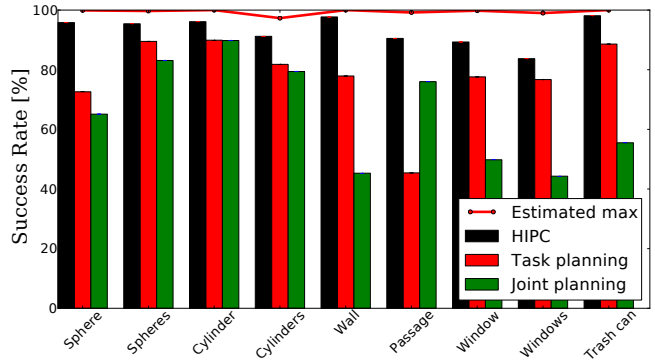


Fig. 7: *Comparison of performance of HIPC and baseline methods in nine types of environments.*

a 77.8% and a 65.4% success rate, respectively. However, from the estimated maximum success rate (Red line), we can see that HIPC did not achieve the maximum success rate. These failures to find a solution could be due to a number of causes, including the 1000s time limit, MPC trajectory tracking errors, and haptic-cost map errors.

In 8 of the 9 environment types tested, the task-space planner outperformed the joint-space planner. In the comparatively simple environments, greedy reaching by the task-space planner saw high success rates. Its ability to snake around obstacles by controlling contact forces allowed it to navigate particularly well through clutter. However, in the **Walls** type clutter, the task-space planner had a low success rate because any of the random initial configurations not aligned with the opening resulted in the arm being trapped outside the walls. The joint-space planner frequently failed due to running out of time, since it easily gets stuck and takes substantial time to replan. For unknown reasons, our controllers result in the real DARCI robot moving much faster than the simulated DARCI robot, which bodes well for future implementations of our system on real robots.

IX. Conclusion

We have introduced our haptically-guided interleaving planning and control framework (HIPC), which interleaves a task-space planner and controller with a joint-space planner and controller that use a shared map of the environment based on accumulated contact information. Both controllers use our previously published dynamic MPC method to follow plans while keeping contact forces low.

Using a physical simulation of a humanoid robot, we evaluated the performance of HIPC in a variety of environments. Over all 9000 trials it had a 93.1% success rate. The worst performance it had in any environment type was an 83.7% success rate over 1000 trials in the **Windows** environment type. HIPC outperformed the two components it interleaves in all 9 types of environments, which demonstrates the value of our approach. We also proved that an idealized version of HIPC is complete for discretized environments.

Acknowledgment*: We thank Youkeun Kim for help with figure generation. This work was supported in part by NIDRR RERC Grant H133E130037, DARPA M3 contract W911NF-11-1-603, and NSF Award IIS-1150157.*

References

[1] J. A. Ambros-Ingerson and S. Steel, "Integrating planning, execution and monitoring.," in *AAAI*, vol. 88, pp. 21–26, 1988.

[2] M. D. Killpack and C. C. Kemp, "Fast reaching in clutter while regulating forces using model predictive control," *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2013.

[3] A. Jain, M. D. Killpack, A. Edsinger, and C. C. Kemp, "Reaching in clutter with whole-arm tactile sensing," *The International Journal of Robotics Research*, p. 0278364912471865, 2013.

[4] I. R. Nourbakhsh, *Interleaving planning and execution for autonomous robots*. Kluwer Academic Publishers, 1997.

[5] C. Park, J. Pan, and D. Manocha, "Real-time optimization-based planning in dynamic environments using gpus," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013.

[6] L. Kaelbling and T. Lozano-Perez, "Hierarchical task and motion planning in the now," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1470–1477, May 2011.

[7] S. Koenig and M. Likhachev, "D* lite.," in *AAAI/IAAI*, 2002.

[8] R. Rusu, I. Sucan, B. Gerkey, S. Chitta, M. Beetz, and L. Kavraki, "Real-time perception-guided motion planning for a personal robot," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 4245–4252, Oct 2009.

[9] D. Um, M. Gutierrez, P. Bustos, and S. Kang, "Simultaneous planning and mapping (spam) for a manipulator by best next move in unknown environments," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 5273–5278, Nov 2013.

[10] A. Petrovskaya and O. Khatib, "Global localization of objects via touch," *Robotics, IEEE Transactions on*, vol. 27, no. 3, pp. 569–585, 2011.

[11] T. Bhattacharjee, P. M. Grice, A. Kapusta, M. D. Killpack, D. Park, and C. C. Kemp, "A robotic system for reaching in dense clutter that integrates model predictive control, learning, haptic mapping, and planning," in *IROS 2014 workshop on Robots in Clutter*, 2014.

[12] J.-C. Latombe, "Robot motion planning, chapter," 1996.

[13] K. Shoemake, "Uniform random rotations," in *Graphics Gems III* (D. Kirk, ed.), pp. 124–132, Academic Press, 1992.

[14] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation (ICRA), 2000 IEEE International Conference on*, pp. 995–1001 vol.2, 2000.

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

[16] T. Bhattacharjee, A. Kapusta, J. M. Rehg, and C. C. Kemp, "Rapid categorization of object properties from incidental contact with a tactile sensing robot arm," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2013.

[17] D. Park, A. Kapusta, Y. K. Kim, J. M. Rehg, and C. C. Kemp, "Learning to reach into the unknown: Selecting initial conditions when reaching in clutter," in *Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on*, 2014.