Automated Planning for Robotic Cleaning Using Multiple Setups and Oscillatory Tool Motions

Ariyan M. Kabir, Krishnanand N. Kaipa, Jeremy Marvel, and Satyandra K. Gupta

***Abstract*— This paper presents planning algorithms for robotic cleaning of stains on nonplanar surfaces. Access to different portions of the stain may require frequent repositioning and reorienting of the object. Some portions with prominent stain may require multiple passes to remove the stain completely. Two robotic arms have been used in the experiments. The object is immobilized with one arm and the cleaning tool is manipulated with the other. The algorithm generates a sequence of reorientation and repositioning moves required to clean the part after analyzing the stain. The plan is generated by accounting for the kinematic constraints of the robot. Our algorithm uses a *depth-first branch-and-bound* search to generate setup plans. Cleaning trajectories are generated and optimal cleaning parameters are selected by the algorithm. We have validated our approach through numerical simulations and robotic cleaning experiments with two KUKA robots.**

***Note to Practitioners*—We encounter nonrepetitive cleaning tasks everyday in both industrial and household environments. Variations in stain pattern, geometry, and material of the object make it difficult to manually program robots for such tasks. In this paper, we present planning algorithms to automate the cleaning task using robots. The practical impact of our approach is evidenced by the actual robot results involving realistic examples like cleaning of hard paint stains on curved surfaces and rust on metal surfaces. Practitioners from industry can use the methods presented in this paper to develop automated robotic systems for nonrepetitive tasks like cleaning and polishing. Our approach caters to the primary requirements of these applications like multiple setups, multiple passes within each setup, and determination of optimal motion parameters like velocity, force, and oscillation frequency of the cleaning tool.**

Manuscript received May 12, 2016; revised October 25, 2016; accepted December 3, 2016. Date of publication March 14, 2017; date of current version June 29, 2017. This paper was recommended for publication by Associate Editors Z. Yin and Prof. M. Wang upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Grant 1200087 and Grant 1634431 and in part by the National Institute of Standards and Technology Cooperative Agreement under Grant 70NANB15H250. Any commercial product or company name in this paper is given for informational purposes only. Their use does not imply recommendation or endorsement by NIST or the University of Southern California or Old Dominion University.

A. M. Kabir and S. K. Gupta are with the Center for Advanced Manufacturing, University of Southern California, Los Angeles, CA 90007 USA (e-mail: guptask@usc.edu).

K. N. Kaipa is with the Department of Mechanical and Aerospace Engineering, Old Dominion University, Norfolk, VA 23529 USA.

J. Marvel is with the National Institute of Standards and Technology, Gaithersburg, MD 20899 USA.

This paper has supplementary downloadable multimedia material available at http://ieeexplore.ieee.org provided by the authors. The Supplementary Materials contains two video files titled "setup_planner_1.mp4" and "setup_planner_2.mp4." The videos demonstrates the working of the methods presented in the paper and physical test results of the setup planner with two robotic manipulators. A plastic bowl and a 3D printed part have been used as target surfaces to clean. Acrylic paint has been used as surrogate for stain. The materials are 16.3 MB and 8.53 MB respectively.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2017.2665460

***Index Terms*— Cleaning by scrubbing, nonrepetitive tasks, robotic cleaning, setup planning.**

I. Introduction

MANY applications like manufacturing, maintenance, service, construction, food processing, and health care involve the common task of cleaning tools and parts. Different cleaning modes may be used, depending on the nature of the cleaning task. Cleaning fluid is utilized, in some cases, to separate foreign particles on the surface by force or to dissolve them. Kitchen utensils and clothing are cleaned using this mode. There are many other cleaning tasks that require mechanical scrubbing with an oscillatory moving cleaning tool. Abrasive particles are usually embedded in the surface of the cleaning tool. Removal of foreign particles involves mechanical erosion in this mode. This paper deals with robotic cleaning with abrasive actions.

Consider a remanufacturing application where the task is to remove rust from a nonplanar surface. The task needs to be handled differently based on factors like patterns of rust, properties of the metal surface, and geometry of the part. Therefore, this is a representative example of nonrepetitive tasks. Typically, this task is performed by humans and is very tedious in nature. If robots can perform this task automatically, then human workers can allocate their time to other tasks where their skills are more needed. A bimanual robotic setup can be useful to reduce the use of fixtures for different parts. One manipulator can be used to immobilize the part and the other can perform the operation. However, it is challenging to automate this seemingly simple task due to the following reasons.

1) It may be difficult to clean the entire stain from one posture. Often, the part must be moved and regrasped to ensure access to the entire part surface.
2) Cleaning progress must be constantly monitored and the plan modified, if needed, to ensure efficient cleaning. This requires a sensor-based feedback loop.
3) Cleaning with mechanical action requires application of force. Often, the parts being cleaned cannot withstand arbitrarily large forces. This requires that the applied force be monitored and controlled to ensure that the part being cleaned is not physically damaged.
4) Cleaning time is considered as a nonvalue-added time in manufacturing applications. Hence, it needs to be minimized by carefully selecting cleaning parameters.
5) Cleaning complex geometries require complex motions.

Past generation industrial robots were not equipped to automatically perform nonrepetitive tasks like cleaning.

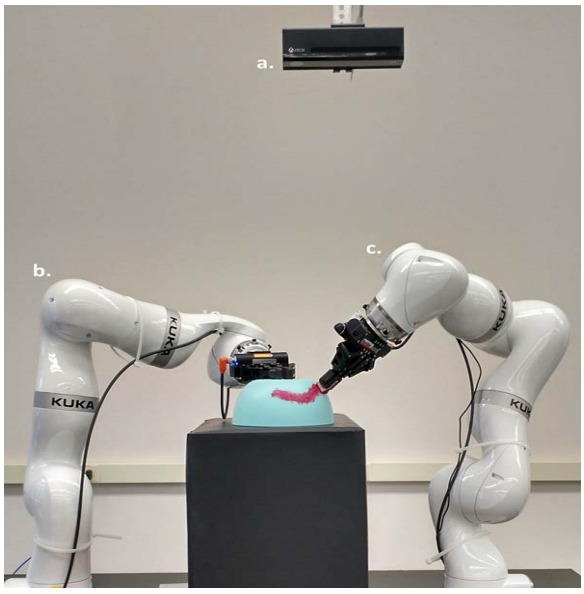

Fig. 1. Robotic setup built with two KUKA robots and Microsoft Kinect. (a) Kinect. (b) Holding robot arm. (c) Cleaning robot arm.

Robotic cleaning has become feasible with recent technological advances. Bimanual robotic setups can be used to immobilize the object to be cleaned with one arm and manipulate the cleaning tool with the other arm. This enables frequent part reorientation without requiring part-specific fixtures, and hence, offers flexibility while performing tasks dealing with a wide variety of geometries. High-resolution torque/force sensors, with support for impedance control, are being integrated into some current robots. These features ensure safe robotic cleaning without causing physical damage to the part.

In this paper, we describe planning algorithms to remove stains from nonplanar objects. Some stain regions may require multiple cleaning passes and the object may need to be repositioned and/or reoriented multiple times. The stain intensity determines the number of required cleaning passes. We used a cleaning tool with an abrasive surface. The experimental setup involves two robot arms. The first arm immobilizes the object. The second arm moves the cleaning tool. Fig. 1 shows the experimental setup used to implement the results of the planning algorithm. The algorithm analyzes the stain and determines the sequence of poses (positions and orientations) needed to clean the part based on the kinematic constraints of the robot arm. Each pose is referred to as a cleaning setup in this paper. Our algorithm generates multipass trajectories for the cleaning tool to follow. Cleaning performance is further optimized through selection of optimal cleaning parameters by our algorithm. The algorithm is capable of refining the plan by observing the cleaning performance.

In our previous work [1], we presented a preliminary approach and studied its feasibility to address the problem of robotic cleaning. This paper builds on our previous work with the following new contributions:

1) improved heuristics based on probabilistic estimation to optimize computation time;
2) analysis of gradient descent approach to improve candidate samples;
3) analysis of different bounds on future cost and their impact on convergence rate and optimality.

In this paper, we propose planning algorithms for robotic cleaning by changing the object's pose based on robot's reachability constraints. Our focus is on problems involving hard stains that typically require multiple passes of mechanical scrubbing over the same surface area and setup change for complete coverage.

II. Related Work

The research community is showing a growing interest in robotic cleaning in recent years. Previous works have approached the problem using perception [2], control [3]–[6], coverage path planning [7], and learning [8]–[10]. Both mobile robots [2] and robot manipulators [3]–[5], [7]–[10] have been utilized in different modes of cleaning. Variation in stain type, geometry of object, surface material, and environmental constraints have led to different approaches. Mobile robots have been used to vacuum dirt like clips, sticks, and paper-balls lying on floors. Robot manipulators have been deployed to clean particles/seeds, talc powder, dry erase marker ink, powdered candy mixed with food color, etc., from flat and curved surfaces. We are interested in cleaning hard stains on curved surfaces that require mechanical scrubbing by robot manipulators. Therefore, we confine our review of prior work to this topic.

Coverage path planning to clean curved surfaces has been addressed by Hess *et al.* [7] using a robot manipulator. They considered different inverse kinematics solutions as graph nodes and grouped the nodes in clusters. They formulated the problem as a generalized traveling salesman problem where the arm needs to visit at least one node in each cluster. They achieved lower completion time and manipulation effort compared to Euclidean coverage algorithms. However, their work concentrated on area coverage and did not consider evaluating cleaning performance.

Sato *et al.* [3] worked on cleaning dry erase marker ink from a white board using a robot manipulator. They developed a trajectory and force tracking controller. Planning for high-level manipulation actions has been presented by Martinez *et al.* [11] where they approached the problem of cleaning dry erase marker on white board and particles/seeds on a flat surface. They used a perception feedback loop to replan actions based on change in dirt distribution. They also presented a learning method to adapt the system with different tool grasp, robot, and surfaces. Equilibrium point control has been used by King *et al.* [4] to introduce wiping motions with relatively low force ($<$3 N) by a robot for a bed bathing application in hospital environments. Powdered candy mixed with food color was used to create stains on a human body in their experiments. The stain was detected by analyzing the hue content with image processing. Imitation learning based on dynamic Bayesian network was used by Eppner *et al.* [9] to develop a two-layer framework that models a human through vision and then uses force profiling to wipe a kitchen table. Dynamic motion primitives was used by Nemec and Ude [10] to wipe a table. In our earlier works [12]–[14], we used semi-supervised learning approaches to learn optimal operation parameters for robotic cleaning.

Efforts on painting, grinding, and polishing are related to robotic cleaning of curved surfaces. Grinding and polishing tasks have considered automated tool path planning on curved surfaces. Ng *et al.* [15] developed a framework to bridge the knowledge transfer gap of the manual operator's skills to a robot program by capturing contact force and tool path with a sensorizedž handheld belt grinder. Zhao *et al.* [16] used a parallel plane slicing method to plan trajectories for robotic blade grinding. Wang *et al.* [17] compared different path patterns for robot-assisted grinding. There have been studies on tool path planning for automotive body painting tasks. Sheng *et al.* [18] developed a method for spray painting complicated, multipatch, sheet-metal surfaces. They partition the surface into feasible patches and generate tool paths to optimize performance. Conner *et al.* [19] considered the effects of surface curvature and deposition pattern to automate tool trajectory generation. Force models for shaving and face wiping have been presented by Hawkins *et al.* [20].

Position and force control methods have been developed by Nagata *et al.* [5] for polishing molds with curved surface. Liu *et al.* [21] used a supervised learning method to learn human welder intelligence for polishing tasks.

III. Problem Formulation

We have adapted the problem formulation that we introduced in our earlier work [1]. We define the task for the robot as cleaning stains on an arbitrary curved surface $\Gamma \in \mathbb{R}^3$. Let $\ell \in \mathbb{R}^6 = \{x, y, z, \alpha, \beta, \gamma\}$ represent a general pose where (x, y, z) and (α, β, γ) represent the position and orientation, respectively, in 3-D. Let $\Gamma(\ell)$ represent the target surface oriented in an arbitrary pose ℓ. We approximate the stain on the surface as a set of small discrete stain patches $\mathcal{P} = \{p_i : i = 1, 2, \ldots, n\}$. Each patch p_i is a small planar triangle with an area $a_i \leq a_m$, where a_m is the surface area of the tip of the cleaning tool. Fig. 2(a) demonstrates an example of a surface after triangulation. The red region represents $\mathcal{P}$.

We assume that the stain intensity is not uniform across the surface and that a single pass may not be able to clean the entire stain region completely. Let N_i represent the number of cleaning passes required to remove the stain from patch p_i. The number of passes is determined by image processing explained in Section VII. We restrict the robot's motion such that its tool axis aligns to the surface normal and the sweeping motions are orthogonal to the surface normal. The robot may fail to satisfy these conditions for some segments of $\mathcal{P}$, for some $\Gamma(\ell)$. For each $\Gamma(\ell)$, we can test how many patches can be reached by the robot by solving its inverse kinematics. This reachability problem can be solved by changing $\Gamma(\ell)$ in steps such that all the subsets of the target surface fall in the robot's reachability space at least once. Therefore, we formulate our cleaning problem as a multisetup, multipass, cleaning task with setup planning for the target surface and trajectory planning for the cleaning robot.

We define a set of candidate setups $\mathcal{S} = \{s_j; j = 1, 2, \ldots, m\}$, where $s_j = \{p_j^i : i = 1, 2, \ldots, k\} \subseteq \mathcal{P}$, $k \leq n$ and the conditions on robot motion is satisfied $\forall p_j^i \in s_j$. Fig. 2(b) and (c) demonstrates two candidate setups to clean $\mathcal{P}$.

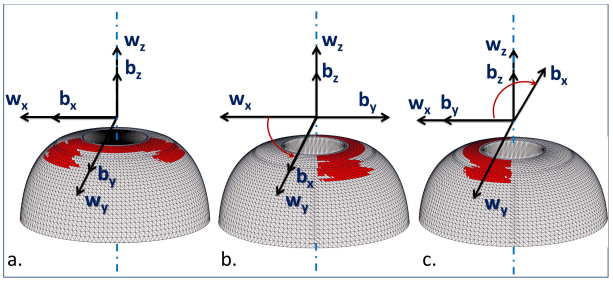

Fig. 2. (a) Representation of an initial setup, i.e., the object's coordinate frame aligned with world coordinate frame. (w_x, w_y, w_z) and (b_x, b_y, b_z) represent the world coordinate frame and the bowl's coordinate frame, respectively. The red region $(\mathcal{P})$ is the target region to clean. (b) and (c) Two sample candidate setups. The setup configurations (b) and (c) are achieved by applying $+90°$ and $-90°$ rotation about w_z axis to the initial setup.

Each setup s_j corresponds to a distinct pose $\Gamma(\ell_j)$. The maximum number of passes to cover s_j is given by $N_j^{\max} = \max_{i=1}^{|s_j|} N_j^i$, where N_j^i is the number of passes to clean p_j^i. Let t_j^i be the time required for the ith cleaning pass for setup s_j. Let, t_j be the time to clean setup s_j, i.e., time to complete $N_j^{\max}$ cleaning passes for setup s_j. Then, $t_j = \sum_{i=1}^{N_j^{\max}} t_j^i$.

Let, t_j^s be the setup time, defined as the time to change the pose of the object from setup s_j to setup s_{j+1}. Let, a_j^i be the surface area of the patch p_j^i in setup s_j. We define the cleaning rate for setup s_j as $R_j = (\sum_{i=1}^{|s_j|} a_j^i)/(t_j + t_j^s)$.

We can generate different ordered setup sequences by permuting $s_i \in \mathcal{S}$. We define a valid setup plan $\mathbb{S} = (s_1, s_2, \ldots, s_q)$, where $q \leq |\mathcal{S}|$, as an ordered sequence of setups that cleans the entire region. The path planner generates a trajectory τ_j for each $s_j \in \mathbb{S}$. The trajectory τ_j comprises $N_j^{\max}$ cleaning passes. The robot may need to reposition the tool to cover the disjointed patches in a setup. Both the cleaning motions and the repositioning motions are captured by τ_j. The sampling based method for generating $\mathcal{S}$ and the algorithm to find setup sequence solutions are described in Sections IV and V-A, respectively. Our method to generate trajectories is described in Section VI. The method to select optimal operation parameters (e.g., tool speed, applied force) is described in Section VIII.

For each setup, s_j, there is an execution time $t^e(\tau_j) = t^c(\tau_j) + t^r(\tau_j)$, where $t^c(\tau_j) = t_j$ is the cleaning time when the tool is in contact with $\mathcal{P}$, while following τ_j and $t^r(\tau_j)$ is the repositioning time when the cleaning robot moves between disjoint patches. We define the total cleaning time for a valid setup plan $\mathbb{S}$ as

$$\mathbb{T}(\mathbb{S}) = \sum_{j=1}^{|\mathbb{S}|} (t^e(\tau_j) + t_j^s). \tag{1}$$

The problem is formally stated as follows: *Find a setup plan* $\mathbb{S}^* = (s_1^*, s_2^*, \ldots, s_k^*), k \leq |\mathcal{S}|$, *where* $s_1^*, s_2^*, \ldots, s_k^* \in \mathcal{S}$, *such that* $\mathcal{P}$ *is completely clean and* $\mathbb{T}(\mathbb{S}^*)$ *is minimized.*

IV. Generating Candidate Setups

A sampling based approach is used to generate the initial set of candidate setups. For each candidate setup s_i, we determine

the number of stain patches that the robot can reach in the desired orientation. This is achieved by solving the inverse kinematics of the manipulator for all the vertices of each patch $p_i \in \mathcal{P}$. We consider a stain patch to be reachable when all three vertices of that patch are reachable by the robot. We then assign that patch to the candidate setup s_i under consideration. While triangulating the surface we pose the constraint that surface area of the patch needs to be smaller than the surface area of the cleaning tool tip.

Our sampling starts with a coarse resolution over a wide range of configuration space parameters. We find a narrower feasible sampling range by eliminating setups that do not cover any patch. We experimented with the following three sampling approaches in the narrower sampling range of configuration parameters: 1) fine resolution uniform sampling; (2) hierarchical uniform sampling with gradient descent; and (3) random sampling with gradient descent. These sampling approaches are described in Sections IV-A1–IV-A3, respectively. The notion of nondominated setups has been used to describe these approaches. We consider a setup s to be dominated if there exists another setup s' that contains all the patches covered by s.

A. Sampling Approaches

1) Fine Resolution Uniform Sampling:

i Initialize an empty set ψ of setups.
ii Set a fine resolution for each axis of the configuration space to perform uniform sampling over the narrower sampling range of configuration parameters.
iii Generate setup samples using the resolution from Step ii and add them to ψ.
iv Eliminate all dominated setups in ψ.
v Send nondominated setups as input to setup planner.

2) Hierarchical Uniform Sampling With Gradient Descent:

i Initialize an empty set ψ of setups.
ii Set a coarse resolution for each axis of the configuration space to perform uniform sampling over the narrower sampling range of configuration parameters.
iii Generate samples of setups using above resolution.
iv Pick setups in the generated set which do not belong to ψ and refine them by using gradient descent over the configuration parameters to optimize the area covered by each setup. Add the refined setups to ψ.
v Eliminate all dominated setups in ψ.
vi If the nondominated setups in ψ do not cover all $p_i \in \mathcal{P}$, then refine sampling resolution and go to Step (iii).
vii If the nondominated setups cover all the patches in $\mathcal{P}$, then send them as input to the setup planner.

3) Random Sampling:

i Initialize an empty set ψ of setups.
ii Generate a random setup sample from the narrower sampling range of configuration parameters.
iii Refine this setup by using gradient descent to optimize the area covered by this setup.
iv If refined setup does not belong to ψ, then add it to ψ.
v Repeat steps ii–iv until setups in ψ cover all $p_i \in \mathcal{P}$.

TABLE I
GRADIENT DESCENT VARIANTS EXPLORED TO IMPROVE AREA COVERED BY A SETUP. SCORING METHODS: TYPE I—CONSERVATIVE; TYPE II—TWO ROUNDS (ROUND 1 IS CONSERVATIVE AND ROUND 2 IS ABSOLUTE)

Abbreviation	Scoring method	Gradient descent type
C_XY	Type I	Along x and y axes for fixed α
CA_XY	Type II	
C_XY_α	Type I	Along x and y axes keeping α fixed and then along α axis alone
CA_XY_α	Type II	
C_XYα	Type I	Along x, y, and α axes
CA_XYα	Type II	

vi Eliminate all dominated configurations in ψ.
vii Send nondominated setups in ψ as input to the planner.

B. Scoring Scheme for Gradient Descent to Improve Area Coverage

We use gradient descent in the configuration space to optimize the area covered by a setup. Let $P_1 \in \mathcal{P}$ be the set of patches that were reachable by the robot for a starting configuration. We define $|P_1|$ as the score for the starting configuration. Suppose one step was taken by gradient descent and it is at a new setup configuration. Let $P_2 \in \mathcal{P}$ be the set of patches that are reachable by the robot for this new configuration. We use two score evaluation schemes:

Conservative Scoring: In this scheme, we enforce a constraint to ensure that the improved configuration is able to cover all the patches that were present in the starting configuration. Therefore, if $P_1 \subseteq P_2$, then the score of the new configuration is $|P_2|$. Else, it is $|P_1 \cap P_2|$.

Absolute Scoring: We do not enforce any constraint. The score for the new configuration is evaluated as $|P_2|$.

We explore different gradient descent vairants by performing the search over the entire configuration space, over the subspaces by batch covering the entire configuration space, with conservative scoring scheme, and with both conservative and absolute scoring schemes (refer to Table I).

V. SETUP PLANNING

The sampling based method described in Section IV leads to a set of refined setups. The setup planner finds a setup plan from this set of candidate setups $\mathcal{S}$.

A. Setup Planner

Our algorithm uses a *depth-first branch-and-bound (DFBnB)* search with computational time bound $\mathcal{T}_{\max}$ to generate setup plans. DFBnB [22] is an efficient search algorithm. We have adopted this algorithm to solve our setup planning problem. Let $s^i_j \in \mathcal{S}$ represent a setup, where j is the node index in the ith solution (not necessarily a sequence of setups that completes cleaning the part). Therefore, the ith solution is given by $\mathbb{S}^i = (s^i_1, s^i_2, \ldots, s^i_k), k \leq |\mathcal{S}|$. The planner implementation consists of the main routine FindSetupPlan($\mathcal{P}, \mathcal{S}$) (Algorithm 1) that calls the routine AddSetUp($\mathcal{P}_r, \mathcal{S}_u, \mathbb{S}_{\text{curr}}$) (Algorithm 2). Cleaning rate is used as a branch-guiding heuristic. At the first instance of a

Algorithm 1: FindSetupPlan $(\mathcal{P}, \mathcal{S})$

1: Initialize $\mathbb{S}^* = \emptyset$
2: Initialize $\mathbb{T}^* = \infty$
3: Call AddSetUp $(\mathcal{P}, \mathcal{S}, \emptyset)$
4: Return $\mathbb{S}^*$

Algorithm 2: AddSetUp$(\mathcal{P}_r, \mathcal{S}_u, \mathbb{S}_{\text{curr}})$

1: If computation time exceeds $\mathcal{T}_{\max}$ then abort search.
2: If $\mathcal{P}_r = \emptyset$ and $\mathbb{T}(\mathbb{S}_{\text{curr}}) > \mathbb{T}^*$ then Return
3: If $\mathcal{P}_r = \emptyset$ and $\mathbb{T}(\mathbb{S}_{\text{curr}}) \leq \mathbb{T}^*$, then
update $\mathbb{S}^* = \mathbb{S}_{\text{curr}}$, $\mathbb{T}^* = \mathbb{T}(\mathbb{S}_{\text{curr}})$ and Return
4: If $\mathbb{T}(\mathbb{S}_{\text{curr}}) + \mathbb{T}^{lb}(\mathcal{P}_r) \geq \mathbb{T}^*$, then Return
5: Otherwise,
If $\exists p \in \mathcal{P}_r$ associated with only one $s \in \mathcal{S}_u$, then
Find $\mathcal{P}(s)$ by patches that are present in s
Call AddSetUp $(\mathcal{P}_r - \mathcal{P}(s), \mathcal{S}_u - s, \mathbb{S}_{\text{curr}} \bigcup s)$
Otherwise,
Sort $\mathcal{S}_u$ by highest to lowest cleaning rate.
For every s in $\mathcal{S}_u$ in decreasing order of cleaning rate
Find $\mathcal{P}(s)$ by patches that are present in s
Call AddSetUp $(\mathcal{P}_r - \mathcal{P}(s), \mathcal{S}_u - s, \mathcal{S}_{\text{curr}} \bigcup s)$

call to Algorithm 2, we compute the cleaning rate (defined in Section III) for all the setups and select the setup $s_1^0 \in \mathcal{S}$, which has the maximum cleaning rate. This results in the remaining stain area $\mathcal{P}_r = \mathcal{P} - \{p_i \in s_1^0\}$ to be covered and the set of un-used setups $\mathcal{S}_u = \mathcal{S} - s_1^0$. Then, in the second instance, we recompute the cleaning rate for all setups in $\mathcal{S}_u$, and select the setup $s_2^0 \in \mathcal{S}_u$, which has the maximum cleaning rate. This cycle repeats by making recursive calls to Algorithm 2 to find other setups s_j^0, one at a time, until all the stain area is covered ($\mathcal{P}_r = \emptyset$). Once the initial solution $\mathbb{S}^0$ is found, we set the best solution $\mathbb{S}^* = \mathbb{S}^0$ and the cleaning time for the best solution $\mathbb{T}^* = \mathbb{T}(\mathbb{S}^0)$, which is the sum of the setup and execution times for all the setups in the solution sequence. Then, we keep branching to find better solutions until $\mathcal{T}_{\max}$ is exceeded.

We use the following branch-pruning heuristic for faster convergence. Let $\mathbb{S}_{\text{curr}}$ represent the set of setups in the current partial solution. We consider $\mathbb{T}(\mathbb{S}_{\text{curr}})$ as the cost of the current partial solution. Next, a lower bound on future cost is defined as the lower bound on the execution and setup time for the remaining stain region $\mathcal{P}_r$

$$\mathbb{T}^{lb}(\mathcal{P}_r) = \min_j (t_j^s + t^e(\tau_j)) \tag{2}$$

where $s_j^i \in \mathbb{S} - \mathbb{S}_{\text{curr}}$. If $\mathbb{T}(\mathbb{S}_{\text{curr}}) + \mathbb{T}^{lb}(\mathcal{P}_r) \geq \mathbb{T}^*$, then we prune that branch from the search tree since it is suboptimal.

B. Setup Planner Using Initial Estimate on Setup Size

In Section III, we defined setup time as the time taken to change setups. Note that no cleaning happens during this time. Therefore, a solution with a large number of setups will lead to low cleaning rate. This implies that the initial solution

Algorithm 3: FindSetupPlan $(\mathcal{P}, \mathcal{S})$

1: Initialize $\mathbb{S}^* = \emptyset$
2: Initialize $\mathbb{T}^* = \infty$
3: IESS $\leftarrow$ InitialEstimateOfSetupSize$(\mathcal{P}, \mathcal{S})$
4: Call AddSetUp $(\mathcal{P}, \mathcal{S}, \emptyset)$
5: Return $\mathbb{S}^*$

Algorithm 4: AddSetUp$(\mathcal{P}_r, \mathcal{S}_u, \mathbb{S}_{\text{curr}})$

1: If computation time exceeds $\mathcal{T}_{\max}$ then abort search.
2: If $|\mathbb{S}_{\text{curr}}| >$ IESS then Return
3: If $\mathcal{P}_r = \emptyset$ and $\mathbb{T}(\mathbb{S}_{\text{curr}}) > \mathbb{T}^*$ then Return
4: If $\mathcal{P}_r = \emptyset$ and $\mathbb{T}(\mathbb{S}_{\text{curr}}) \leq \mathbb{T}^*$, then
update $\mathbb{S}^* = \mathbb{S}_{\text{curr}}$, $\mathbb{T}^* = \mathbb{T}(\mathbb{S}_{\text{curr}})$ and Return
5: If $\mathbb{T}(\mathbb{S}_{\text{curr}}) + \mathbb{T}^b(\mathcal{P}_r) + \mathbb{T}_{\text{decay}} \geq \mathbb{T}^*$, then Return
6: Otherwise, If $\exists p \in \mathcal{P}_r$ associated with only one $s \in \mathcal{S}_u$, then
Find $\mathcal{P}(s)$ by patches that are present in s
Call AddSetUp $(\mathcal{P}_r - \mathcal{P}(s), \mathcal{S}_u - s, \mathbb{S}_{\text{curr}} \bigcup s)$
Otherwise,
Sort $\mathcal{S}_u$ by highest to lowest cleaning rate.
For every s in $\mathcal{S}_u$ in decreasing order of cleaning rate
Find $\mathcal{P}(s)$ by patches that are present in s
Call AddSetUp $(\mathcal{P}_r - \mathcal{P}(s), \mathcal{S}_u - s, \mathcal{S}_{\text{curr}} \bigcup s)$

produced by the setup planner may be suboptimal. The number of setups in the optimal solution may be much lower than that in the initial solution. Since the algorithm considers the total time of this suboptimal solution as an initial bound and keep updating when a better solution is found, it will take a long time to converge. However, the convergence will be faster if we could use a better initial bound. If we can estimate the setup size in the optimal solution (i.e., the number of setups in the optimal solution or the size of the minimum set cover)[1] before running the AddSetUp algorithm, then we could use it as a branch-pruning heuristic to converge faster towards the optimal solution.

For this purpose, we modified Algorithms 1 and 2 as Algorithms 3 and 4, respectively. Algorithm 3 calls InitialEstimateOfSetupSize$(\mathcal{P}, \mathcal{S})$ (Algorithm 5) in step 3, which gives us an initial estimate of the setup size (IESS) of the optimal solution. step 2 of Algorithm 4 prunes the branches in the search tree when the solution size exceeds the initial estimate (IESS). Therefore, it makes the search converge faster towards the solution compared to Algorithm 2.

Algorithms 2 and 4 are depth-first branch-and-bound search algorithms. They are constructed in a recursive manner. Their computational complexity is exponential in time with order of $|\mathcal{S}|$, where $\mathcal{S}$ is the set of candidate setups. The exponent is the number of nodes in the first solution branch. In Algorithm 5, N_{repeat} is the number of times the random sampling process is repeated. Algorithm 5 is constructed as two nested loops. The outer loop runs N_{repeat} times. The inner loop picks one setup from the set of candidate setups ($\mathcal{S}$) in

[1] Subcollection of the setups which will cover all stain patches.

Algorithm 5: InitialEstimateOfSetupSize($\mathcal{P}, \mathcal{S}$)

```
Initialize ψ = ∅
Initialize N = [ ]
For i in range(N_repeat)
    Initialize ℙ = 𝒫
    While( |ℙ| > 0 )
        Randomly pick s ∈ 𝒮 − ψ
        ψ ← ψ ∪ s
        Find 𝒫(s) by patches that are present in s
        ℙ ← ℙ − 𝒫(s)
    N.append(|ψ|)
Return min(N)
```

each iteration until $|\mathbb{P}|$ is empty. Therefore the inner loop runs $|\mathcal{S}|$ times or lower. However, as explained in Section IX-B, N_{repeat} needs to be significantly larger than $|\mathcal{S}|$. Therefore, the computational complexity of Algorithm 5 is linear in time with N_{repeat}.

1) Initial Estimate on Setup Size: Algorithm 5 generates the initial estimate on number of setups by finding the set cover through random sampling. It repeats this process for N_{repeat} times. The probability of finding the minimum set cover at least once in N_{repeat} trials can be be analytically derived as follows.

Let $\mathbb{S}_c = \{\mathcal{S}_c^i \subseteq \mathcal{S}\}$ represent the set of all set-covers. The minimum set-cover $\mathcal{S}_c^* \in \mathbb{S}_c$ is given by

$$\mathcal{S}_c^* = \{s_1^*, \ldots, s_q^* \in \mathcal{S} \mid \bigcup_{i=1}^{q} \mathcal{P}(s_i^*) = \mathcal{P}\} \tag{3}$$

where

$$q = arg\min |\mathcal{S}_i|, \quad \mathcal{S}_i \in \mathbb{S}_c. \tag{4}$$

Note that if $|\mathcal{S}_c^*| = |\mathcal{S}|$, then the size of minimum set-cover can be found using Algorithm 5 with probability one.

Next, assume that the size of the minimum set-cover $q = |\mathcal{S}_c^*| < |\mathcal{S}|$ and that $\mathcal{S}_c^*$ is unique. The probability of finding this unique set cover is given by the following theorem.

Theorem: If there is a unique minimum set cover $\mathcal{S}_c^* \subset \mathcal{S}$ such that $q = |\mathcal{S}_c^*| < m = |\mathcal{S}|$, then the probability of finding $\mathcal{S}_c^*$ using Algorithm 5 is given by

$$1 - \left(1 - \frac{q! \times (m-q)!}{m!}\right)^{N_{\text{repeat}}}. \tag{5}$$

Proof: Algorithm 5 (steps 5−9) finds a set cover by randomly sampling one setup at a time from $\mathcal{S}$.
Since $q < m$, the probability of finding the minimum set-cover in one trial is given by

$$\begin{aligned} p &= \left(\frac{q}{m}\right)\left(\frac{q-1}{m-1}\right)\cdots\left(\frac{q-(q-1)}{m-(q-1)}\right) \\ &= \frac{q! \times (m-q)!}{m!}. \end{aligned} \tag{6}$$

Hence, the probability of not finding the minimum set cover in one trial is $(1-p)$. This implies that the probability of not finding the minimum set cover by repeating the random sampling process for N_{repeat} times is $(1-p)^{N_{\text{repeat}}}$.

Therefore, the probability of finding minimum set cover atleast once in N_{repeat} trials is given by

$$1 - (1-p)^{N_{\text{repeat}}} = 1 - \left(1 - \frac{q! \times (m-q)!}{m!}\right)^{N_{\text{repeat}}}.$$

The above theorem considered the case when the minimum set-cover is unique. However, there could be multiple minimum set-covers of the same size. In these cases, the probability will be much higher, and (5) gives us a lower bound on the probability of finding a minimum set cover using Algorithm 5. Note that, we cannot predict q beforehand. We can numerically evaluate the probability given by (5) for $q = 1, 2, \ldots, m-1$ with different N_{repeat}. This can guide us to select the best N_{repeat} for different m. In Section IX-B, we discuss how to choose N_{repeat} such that Algorithm 5 can guarantee the size of optimal solution with high probability.

The heuristic based on IESS does not prune branches with optimal solution if the setup size of the true optimal solution size is lower than IESS. This is because it only prunes branches with solution size higher than IESS. Suppose the size of the optimal solution is IESS. There might be multiple solutions of the same size but with different cleaning rates. Since branches with solution size $\leq$ IESS are not pruned, the search will still yield a solution with optimal cleaning rate.

The effectiveness of the branch-pruning heuristic in step 4 of Algorithm 2 depends on the setup time. It may generate a solution with a large number of setups if the setup time is too low. The initial estimate on the setup bound heuristic makes Algorithm 4 robust against variations in setup time. It will guarantee a minimum setup size with a good probability even if it fails to guarantee a solution with true optimal cleaning rate.

2) Different Bounds on Future Cost: Step 4 of Algorithm 2 and step 5 of Algorithm 4 are branch-pruning heuristics that prune branches based on future cost of covering remaining patches. In Section V-A, the lower bound on future cost was given by (2). If we can guarantee with high enough probability that Algorithm 5 can estimate the solution size of the minimum set cover, then we can modify (2) as follows:

$$\mathbb{T}^b(\mathcal{P}_r) = \max_i \left(t_i^s + t^e(\tau_i)\right) \quad \text{where} \quad s_i \notin \mathbb{S}_{\text{curr}}. \tag{7}$$

This will enable the search to converge faster. However, this will guarantee an optimal solution with a probability ≤ 1.0.

The algorithm may find the least setup-size solution very fast (might not be optimal in cleaning rate). However, it may still take a long time to prune all the remaining branches when the number of nondominated setups or setup time is high. In step 5 of Algorithm 4, the term $\mathbb{T}_{\text{decay}}$ on the left side of the inequality, can be an exponential function of the number of nodes expanded in the search tree. This will accelerate convergence (similar to simulated annealing Algorithm) for practical purposes, when the true optimal solution might not be absolutely desired. In our experiments, we set $\mathbb{T}_{\text{decay}} = 0$ and consider the setup time to be fixed for all setups.

VI. Trajectory Planning

A cleaning trajectory is generated for each pass of each setup by creating a spline through the connected triangles

in the setup. Therefore, the cleaning trajectory consists of multiple splines for a single setup. A setup $s_j^* \in \mathbb{S}^*$ may require $N_j^{\max}$ passes to complete cleaning. Correspondingly, there are a total $N_j^{\max}$ number of trajectories generated for the setup s_j^*. For example, if $s_j = \{p_1^j, p_2^j, p_3^j\}$ and the number of passes required for p_1^j, p_2^j and p_3^j are 1, 2, and 3, respectively, then the first trajectory will pass through all the patches p_1^j, p_2^j, and p_3^j. The second trajectory will pass through p_2^j and p_3^j, and the third trajectory will cover p_3^j alone. This strategy utilizes the advantage of continuous motion and saves cleaning time. We find most stains to be locally continuous in reality. Therefore, all the stain patches in a locally continuous stain region may require the same number of cleaning passes. It will take more time if the robot stops at each stain patch to complete all the required cleaning passes for it and then moves to the next patch.

The target surface is intersected with equally spaced parallel planes to generate the continuous trajectory [23], [24]. We sample the curves resulting from the intersections to get waypoints. From the waypoints, we create spline curves as nominal trajectories to be followed by the cleaning tool.

An example of a cleaning trajectory is illustrated in Fig. 3(a). If the robot fails to travel through all the desired points on a single spline, we segment the trajectory into multiple, small, splines. The robot arm then repositions itself suitably at the starting point of each small spline such that it can follow the entire trajectory. The arm is also repositioned to avoid traveling through stain-free regions that may lie between remaining stain regions. Repositioning is equivalent to jumps as the arm comes out of contact with $\mathcal{P}$ during these moves, hence incurring repositioning time. The repositioning trajectory is generated as a spline that connects the current pose of the tool, an intermediate pose where tool is not in contact with $\mathcal{P}$, and the starting pose of the tool for the next cleaning spline.

We overlay a force oscillation on top of the nominal trajectory to expedite cleaning. This is described in Section VIII. Fig. 3(b) illustrates an example of a trajectory with overlaid force oscillation. In our implementation, we used the spline motion primitive of the KUKA Robotics Application Programming Interface. It creates smooth spline curves for the cleaning tool by taking a set of reachable waypoints as input. We denote the trajectory for the setup s_j by $\tau_j = \{\tau_j^{cl}, \tau_j^{\text{repo}}\}$, where τ_j^{repo} is the repositioning trajectory and τ_j^{cl} is the cleaning trajectory.

VII. Perception

We need object and stain detection methods to automate the robotic cleaning process. We used the iterative closest point (ICP) algorithm [25] to match the computer-aided design (CAD) model of the object to its point cloud captured by an RGB-D camera. We recognize the stain on the object by using image processing. The required number of cleaning passes for each triangulated stain patch is estimated at the beginning. The cleaning performance is evaluated after each cleaning pass based on how much stain is removed. After completing all the cleaning passes, we evaluate the cleaning rate.

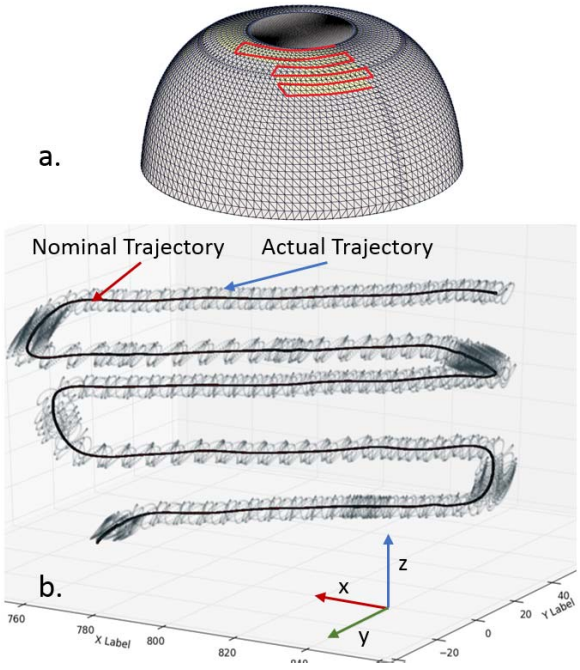

Fig. 3. (a) Example of a trajectory generated by parallel plane slicing for a setup. (b) Overlay pattern generated by an example set of parameters. The values for the parameters used in this example are $f_x = 6$ Hz, κ_x, κ_y =5000 N/m, $A_x, A_y = 30$ N, $\upsilon = 0.3$, and $f_n = 10$ N.

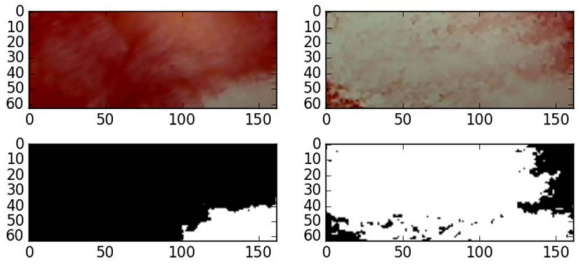

Fig. 4. Clockwise from top-left: before cleaning (original), after cleaning (original), after cleaning (binary), and before cleaning (binary).

K-means clustering was used to classify the stain and background color of the object from the red-green-blue (RGB) image. This vector quantization method considers the color of each pixel as a vector in 3-D space of RGB colors. We set the clustering algorithm to classify the pixels in two colors (binary representation). Two examples of stain detection are illustrated in Fig. 4. White and black colors represent the cleaned area and remaining stains, respectively.

Let N_p^- and N_p^+ be the number of black pixels in the binary image of the target surface before and after a cleaning pass, respectively. Let I_h and I_w be the height and width of the target region in pixels. We define cleaning performance for a single pass as $C_p = (N_p^- - N_p^+)/(I_h \times I_w)$.

Our setup planner needs to know the number of cleaning passes required to clean each triangulated stain patch. With the K-means algorithm, we can classify the pixel colors of the stain image into N_c number of color clusters. If we have a rich database of stain images, then we can use supervised learning methods to determine N_c. In our current implementation,

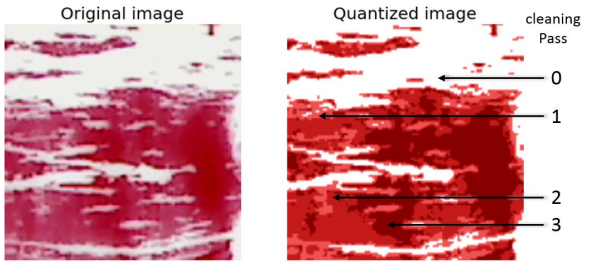

Fig. 5. Pixelwise vector quantization of stain image.

N_c is user defined. The norm of the RGB color value of the clusters is used to determine the number of cleaning passes required for that particular color. An example for $N_c = 4$ is illustrated in Fig. 5. The RGB color value with the largest norm represents zero cleaning pass and the color with the lowest norm represents three cleaning passes. This convention will flip if the stain color is relatively light with respect to the background color. Suitable multiplication factors may also be applicable to the number of cleaning passes determined by this approach.

VIII. Operation Parameter Selection

Mechanical scrubbing is necessary in many practical applications to remove hard stains from a surface. The scrubbing action can be characterized as an oscillation on the cleaning tool. The nominal tool trajectory was defined as splines in our method. In our preliminary experiments, we observed that oscillations along and normal to the direction of tool frame gives better cleaning compared to oscillation in only one direction. Therefore, the Lissajous pattern was selected for overlaid force oscillation on top of the nominal trajectory to expedite cleaning. The force oscillation was overlaid on the XY-plane of the tool frame, whereas the cleaning force was applied to the part surface along the z-axis of the tool frame. The frequency (f_x) and amplitude (A_x) of the force oscillation were controlled along the x-axis of the tool frame. Along the y-axis of the tool frame, the frequency and amplitude were $f_y = 0.4 \times f_x$ and $A_y = A_x$. The phase offset between the x- and y-axes was $1/2 \times \pi$.

In our physical experiments, we explored the effect of the following robot motion parameters on the cleaning performance: 1) robot tool speed v; 2) force f_n applied by the tool in the orthogonal direction to the surface; (3) stiffnesses κ_x and κ_y in the x- and y-directions of the tool reference frame; (4) frequency f_x (or f_y) of the forced oscillation in the x- (or y)-direction with respect to the tool reference frame; and (5) amplitude of overlaid force A_x (or A_y) for Lissajous force oscillation mode.

It is difficult to represent cleaning performance as a closed-form function of these parameters. We adapted the Gaussian process regression (GPR)-based semi-supervised learning method developed in our previous work [12] to estimate the optimal values for these parameters to achieve the desired performance.

We started by defining the task performance target. We wanted the cleaning performance to meet or exceed a certain given threshold (C_p^{th}). Our objective function was a known function designed to maximize tool speed and minimize applied force. However, the model of cleaning performance is not known, therefore we considered the cleaning performance to be a black-box inequality constraint. We then defined the operation parameter limits. The minimum and maximum values of the operation parameters were selected based on the safety considerations and the robot capabilities. We uniformly sampled the parameter space at a coarse resolution to conduct initial exploratory experiments. These experimental data were used to fit a surrogate (Gaussian Process) model for the cleaning performance. Our parameter selection algorithm identifies candidate points from which the next point for experiment is to be selected. The algorithm evaluates the objective function at these candidate points. It also estimates the probability of the points to meet the cleaning performance constraint using GPR. The algorithm identifies the points that meet the probability threshold (Pth) for meeting cleaning performance constraint. From these points, the algorithm returns the point, which minimizes the objective function to conduct a new physical experiment. We call this an exploitation experiment. The surrogate model is updated at each iteration with new experimental data. The iterative algorithm repeats exploitation experiments until it converges to a solution.

IX. Results

A. Synthetic Test Cases

We conducted experiments in simulation with a Puma 560 robot to evaluate our setup planner. We considered four synthetic test objects, as illustrated in Fig. 6. They are represented as triangulated meshes. The area of each triangular face is less than 1.5 cm^2. The red regions represent the stain and consist of about 4000 to 5000 stain patches on each object. Stain intensity was user defined such that not more than three cleaning passes would be required. In our experiments we restricted ourselves to a 3-D configuration space (x, y, and α, where α is the rotation about z) for the objects' pose (i.e., we did not consider roll, pitch, and translation about z-axis). The narrowed down sampling region in this space was [0, 1 m] $\times$ [0, 1 m] $\times$ [$0, 2\pi$].

1) Baseline Results: We conducted fine resolution sampling on the configuration space, removed dominated setups, and ran the setup planner on the nondominated setups to find the number of lowest possible setups for each synthetic test object. We did not use gradient descent in these baseline experiments. We found that the largest sampling resolution corresponding to the solution with minimum number of setups was 50 mm $\times$ 50 mm $\times$ 10° for all the test cases. This leads to 16 317 setup configurations to start with. Table II summarizes the baseline results. We have a reachability test function that solves the inverse kinematics of the robot for each stain patch for a given setup. Each instance of this function takes a significant amount of computational time. Let n_{rfc} be the number of calls made to the reachability test function. For example, let us assume there are 5000 stain patches on a target surface. For each pose of the object, we need to test if the robot can reach all the three vertices of each of the stain patches.

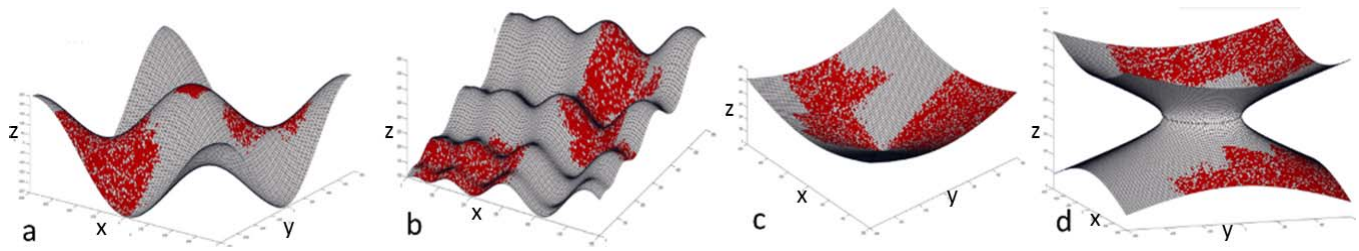

Fig. 6. Four curved surfaces used as synthetic test objects. (a) Sine function (1600 mm × 1600 mm × 500 mm). (b) Schwefel function (600 mm × 600 mm × 350 mm). (c) Concave bowl (1200 mm × 1200 mm × 530 mm). (d) Hyperboloid of one sheet (1200 mm × 1200 mm × 412 mm).

TABLE II
BASELINE RESULTS FOR THE FOUR TEST CASES IN SIMULATION

Case	No. of setups	Non Dominated Setups
Sine	4	36
Schwefel	1	1
Conic	5	531
Hyperboloid	4	751

For each pose of the object, this will involve finding the inverse kinematics solution of the robot for 15 000 times. If there are 12 candidate setups, then we will need to solve inverse kinematics for 180 000 times. Therefore, we use the number of calls to the reachability test function (n_{rfc}) as a performance metric of the algorithms. Lower n_{rfc} with optimal setup plan indicates better performance of the algorithm. For all the cases in Table II, $n_{\text{rfc}} = 16{,}317$.

2) Comparison Between Uniform and Random Sampling Approaches With and Without Gradient Descent: As mentioned in Sections IV-A2 and IV-A3, we experimented with hierarchical uniform sampling and random sampling approaches. Different gradient descent variants (Section IV) were added on top of these sampling schemes and their performance was evaluated. Different sampling resolutions were used at different hierarchy levels for the hierarchical sampling approach. Sampling resolution was scaled by half along one axis at a time at the different hierarchies. Table III summarizes the results of hierarchical uniform sampling. Optimal setup solution was found for all four cases by refining the set of candidate setups with the CA_XY variant of gradient descent (refer to Table I).

We evaluated the random sampling approach by repeating the experiment 100 times for each test case with each type of gradient descent. Table IV shows the likelihood of finding the optimal solution for the synthetic objects. The CA_XY variant worked reasonably well for all four test cases. The number of function calls for the Sine-object, Schwefel-object, Concave bowl, and Hyperboloid-object is illustrated in Fig. 7. We can see that sampling without gradient descent, and sampling with C_XY and CA_XY gradient descent have relatively low variation in n_{rfc}.

From the experiments on synthetic cases, we found that hierarchical uniform sampling with the CA_XY gradient descent variant guarantees optimal solution. The random sampling based approach has significant reduction in n_{rfc}. However, it guarantees an optimal solution with a probability less than 0.5.

TABLE III
RESULTS OF HIERARCHICAL UNIFORM SAMPLING FOR FOUR SYNTHETIC OBJECTS

Case	Gradient Descent Type	No. of setups in solution	Resolution level to find set-cover	n_{rfc}
Schwefel	W/o Grad. Dsc.	1	1	36
	CA_XY	1	1	264
	CA_XY_α	1	1	811
	CA_XYα	1	1	426
Sine	W/o Grad. Dsc.	5	1	36
	CA_XY	4	1	536
	CA_XY_α	5	1	1074
	CA_XYα	5	1	659
Concave bowl	W/o Grad. Dsc.	6	1	36
	CA_XY	5	1	574
	CA_XY_α	5	1	1179
	CA_XYα	7	2	1209
Hyperboloid	W/o Grad. Dsc.	5	1	36
	CA_XY	4	1	457
	CA_XY_α	4	1	1089
	CA_XYα	5	1	671

TABLE IV
LIKELIHOOD OF FINDING OPTIMAL SOLUTION BY RANDOM SAMPLING

Grad. Dsc. Type	Sine	Schwefel	Concave bowl	Hyperboloid
Without Gradient Descent	0	0.13	0.01	0.05
C_XY	0.1	0.35	0.14	0.16
CA_XY	0.19	0.35	0.21	0.32
C_XY_α	0.02	0.51	0.02	0
CA_XY_α	0.07	0.54	0.04	0.01
C_XYα	0.03	0.41	0	0
CA_XYα	0.05	0.43	0.05	0.08

Fig. 8 represents the landscape of area coverage by absolute scoring on a uniformly sampled grid on the $x - y$ plane for a fixed α. For any (x, y) location of the object, the $z - axis$ corresponds to the number of triangles reachable by the robot. The sampling resolution was 50 mm in both the x- and y-directions. The black-star represents the randomly selected initial seed (x, y configuration) and number of triangles covered by this configuration. The green-star and the red-star represent points indicating the number of triangles covered by the configurations, after running a gradient descent using conservative and absolute scoring schemes, respectively. We can see that both scoring schemes give significant

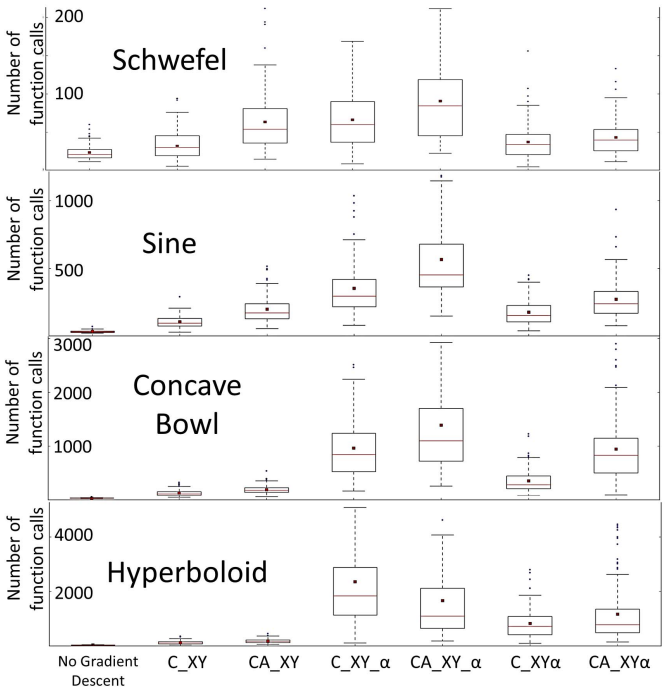

Fig. 7. Variation in n_{rfc} for different kinds of gradient descent in the random sampling-based approach.

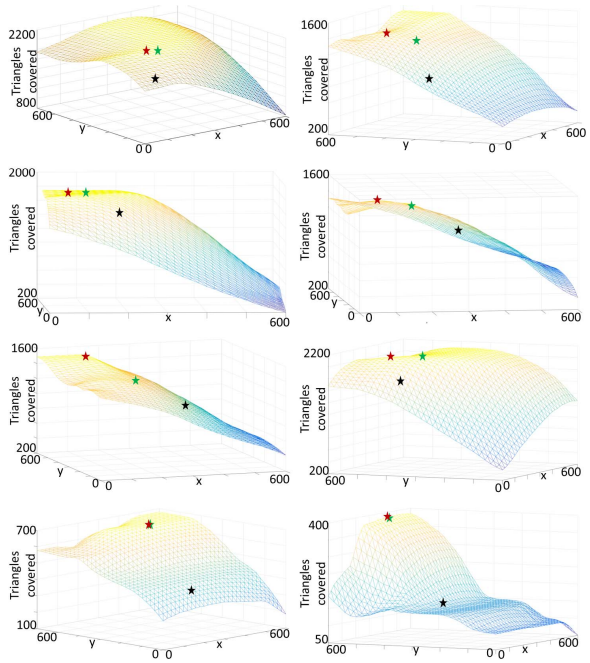

Fig. 8. Eight examples of gradient descent along x & y keeping α fixed (CA_XY type gradient descent) for the convex bowl used in physical test. In these figures, the x- and y-axes represent the (x,y) configuration of the part. The z-axis represents number of reachable stain patches on the part by the robot, at the corresponding (x, y) location of the part. Therefore, the mesh is representing the landscape of the number of reachable stain patches for different (x, y) configuration of the part. The number of stain patches reachable by the robot at the initial (x, y) location of part is represented with black-star. The green-star and red-star represent the number of stain patches reachable by the robot at the new (x, y) location after applying gradient descent using conservative and absolute scoring, respectively.

improvement in area coverage for gradient descent on the $x-y$ plane keeping α fixed.

3) Performance of CA_XY Type Gradient Descent: Intuitively, we can understand that changing orientation can make abrupt change in the number of reachable stain patches. Whereas, changing position makes gradual change in the number of reachable stain patches. This phenomenon is reflected in the landscapes in Fig. 8, which explains why CA_XY type gradient descent performed well for both hierarchical uniform sampling and random sampling approaches.

4) Comparison of Heuristics for Setup Planner: We used a branch-guiding heuristic and a branch-pruning heuristic in our depth-first-branch-and-bound-search. The branch-guiding heuristic is based on cleaning rate. In step 5 of Algorithm 2 and step 6 of Algorithm 4, the candidate setups are sorted based on cleaning rate. The setup with relatively high cleaning rate is chosen as a potential optimal solution branch. The branch-pruning heuristic is based on bound on future cost (future cleaning time). Step 4 in Algorithm 2 and step 5 in Algorithm 4 prune branches based on the estimated bound on future cost.

We experimented by keeping either of these heuristics ON or OFF to see their impact on convergence rate of the search. Table V summarizes the results for three synthetic objects. In these experiments, there were 36 candidate setups. They were uniformly sampled without refinement using gradient descent. There were 7, 12, and 13 nondominated setups for Hyperboloid, Sine, and Concave bowl objects, respectively. The number of setups in the optimal solution were 5, 5, and 7 for Hyperboloid, Sine, and Concave bowl objects, respectively (established using the method described in Section IX-A1).

TABLE V
COMPARISON OF HEURISTICS FOR SETUP PLANNER

Object	Branch Pruning Heuristic	Branch Guiding Heuristic	Node expansion in search tree before convergence	Percentage of node expansion w.r.t. OFF-OFF
Hyperboloid	OFF	OFF	3138	100 %
	OFF	ON	3104	98.92 %
	ON	OFF	679	21.64 %
	ON	ON	569	18.13 %
Sine	OFF	OFF	1188175	100 %
	OFF	ON	1176109	98.98 %
	ON	OFF	39822	3.35 %
	ON	ON	20286	1.71 %
Concave Bowl	OFF	OFF	16260590	100 %
	OFF	ON	16242862	99.89 %
	ON	OFF	362208	2.23 %
	ON	ON	353630	2.17 %

The branch pruning heuristic leads to faster computation time by terminating infeasible branches in the search tree. The impact of the branch pruning heuristic can be seen in Table V. It significantly reduced the number of node expansions in the search tree. On the other hand, the branch guiding heuristic is crafted to help the search algorithm find setups with higher cleaning rates first. This helps to find a feasible branch faster. Keeping both the heuristics active leads to quickest convergence in search.

TABLE VI
COMPARISON OF ALGORITHMS 2 AND 4 WITH TWO DIFFERENT BOUNDS ON FUTURE COST

Bound on Future Cleaning Time			$\mathbb{T}^{lb}(\mathcal{P}_r)$			$\mathbb{T}^{b}(\mathcal{P}_r)$			$\mathbb{T}^{b}(\mathcal{P}_r)$		
AddSetUp Algorithm			Algorithm 2			Algorithm 2			Algorithm 4		
	Non Dominated Setups	Setup Time (s)	Node Expansion	Solution Size	Time (h:m:s)	Node Expansion	Solution Size	Time (h:m:s)	Node Expansion	Solution Size	Time (h:m:s)
Hyperboloid	7	0	513	5	00:00:40	132	7	00:00:26	67	5	00:00:22
		60	569	5	00:00:40	168	6	00:00:27	234	5	00:00:25
		3000	1327	5	00:01:02	1067	5	00:00:53	1101	5	00:00:54
Sine	12	0	12531	9	00:07:55	229	11	00:00:30	2377	5	00:00:57
		60	20286	6	00:12:53	2200	5	00:01:07	2799	5	00:01:10
		3000	97124	6	00:47:57	85795	5	00:41:51	85795	5	00:42:33
Concave Bowl	13	0	353630	7	03:04:05	108	10	00:00:31	2157	7	00:01:19
		60	484371	7	03:35:15	3110	7	00:02:49	25677	7	00:09:12
		3000	1283388	7	06:19:42	1002333	7	04:37:58	1135870	7	04:12:10

B. Comparison of Algorithm 2 *and Algorithm* 4 *With Two Different Bounds on Future Cost*

Table VI summarizes the performances of Algorithms 2 and 4 under different setup times with two different bounds on future costs. In these experiments, candidate setups were generated using uniform sampling without gradient descent. The computation was performed on a single core of a machine with an Intel Xeon E3-1241 3.50 GHz processor and 8GB RAM. We see that larger setup time leads to higher node expansions, hence slower convergence. This is because when setup time is much greater than the execution time, then the future cost based branch-pruning heuristic becomes less effective. As described in Section V-B1, we also see that the performance of Algorithm 2 varies with setup time. We see that it is more likely to get suboptimal solutions with lower setup time. Also, setting $\mathbb{T}^b(\mathcal{P}_r)$ using (7) leads to faster convergence over using (2). This can guarantee optimality with probability ≤ 1.0 for proper choice of N_{repeat}. We can see that the combination of Algorithm 4 and (7) can lead us to an optimal solution with a good probability at a much faster rate.

Fig. 9 guides us in selecting N_{repeat} such that Algorithm 5 guarantees optimality with probability ≥ 0.90. Since we do not know the size of minimum set cover q beforehand, we need to compute the probability for different q for different N_{repeat}. If the cardinality of the set of candidate setups $m = |\mathcal{S}|$, then we need to numerically test the probability for $q = 1, 2, \ldots, m-1$ for different values of N_{repeat}.

Suppose we want Algorithm 5 to give us a correct estimate with probability 0.90. Then, for each m and q, we can evaluate the probability using (5) by gradually increasing N_{repeat} and stopping when probability $\geq$0.90 is reached. We should pick the highest N_{repeat} that we found among different q for a fixed m. From Fig. 9(a)–(d), we can see that N_{repeat} vs q forms a bell curve and N_{repeat} is highest for q close to $m/2$. From Fig. 9(e)–(h), we can see for what values of N_{repeat} the probability converges to 1.0. The fastest rising curves in Fig. 9(e)–(h) are for $q = 1$ *and* $m-1$, and the slowest rising curves are for $q = m/2$. Therefore, for a given m we can evaluate (5) for $q = \lceil m/2$ by gradually increasing N_{repeat} and stopping when our desired probability threshold is reached.

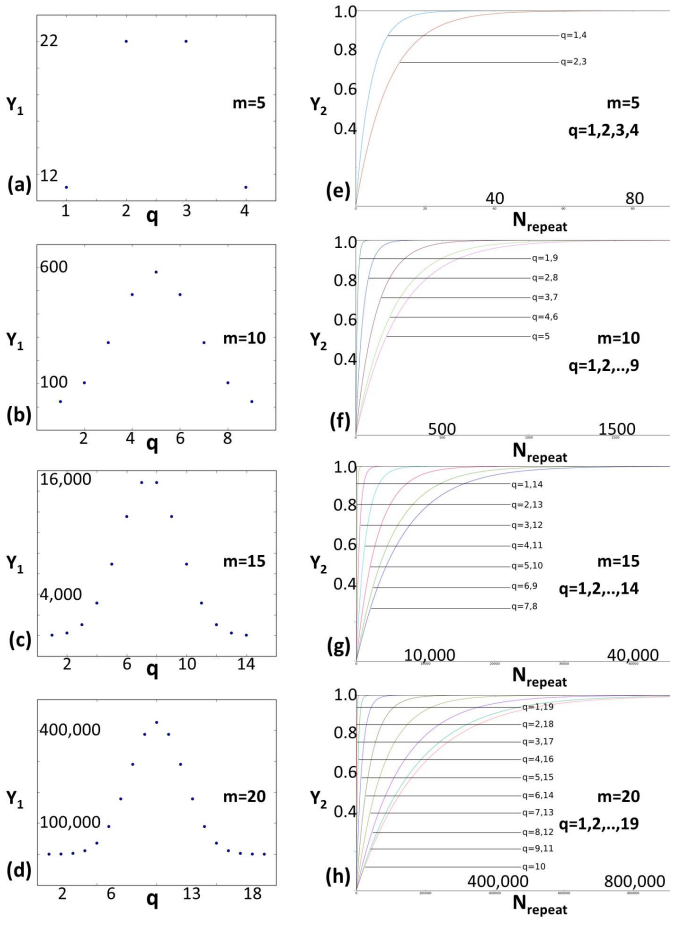

Fig. 9. (a)–(d) (Y_1) vs q, where $Y_1 = N_{\text{repeat}}$ to find minimum set cover size with probability 0.90. (e)–(h) (Y_2) versus N_{repeat}, where Y_2 = probability of finding minimum set cover size. $m = |\mathcal{S}|$ =cardinality of the set of candidate setups, for q = possible minimum set cover size = $1, 2, \ldots, m-1$. In (e)–(h), each curve represents different q.

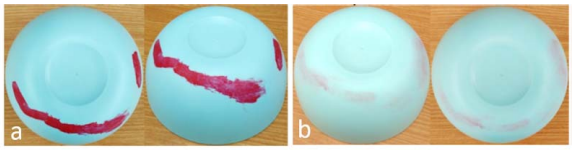

Fig. 10. Bowl surface (a) before cleaning and (b) after cleaning. Images taken from two different angles.

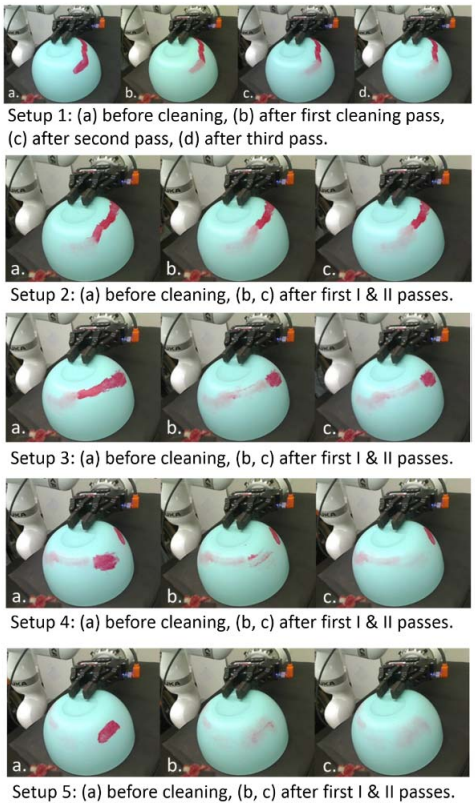

Fig. 11. Snapshots from a video showing execution of cleaning on the plastic bowl according to the setup plan comprising five setups. The part's surface is mostly convex.

C. Physical Testing Results

We conducted robotic cleaning experiments with two KUKA iiwa 7 manipulators. One robot immobilized or held the part and the other robot manipulated a cleaning tool. We used a plastic bowl and a 3-D printed model of ship hull as curved surfaces. The bowl had convex surface. The ship hull model had convex, concave, and flat regions on the surface. A human operator performed the setup changes manually for the bowl. The ship hull model was mounted on one robot arm to perform setup change automatically. We applied acrylic paint and mud on them as a surrogate for stains as shown in Figs. 10 and 12, respectively. We applied different layers of stain on different regions to obtain nonuniform stain intensity. The bowl and hull surfaces were represented as a mesh of 15,644 and 22,764 triangles. The stain region consisted of 1000 and 5709 triangles. Each triangle had a surface area of less than 10 mm^2. The dimensions of the bowl were 21.5 mm $\times$ 21.5 mm $\times$ 9.5mm. The dimensions of the hull model were 20.1 mm $\times$ 18.0 mm $\times$ 14.0mm. For the bowl, we used the following values for the robot motion parameters: 1) $f_x = 7$ Hz; 2) $\kappa_x, \kappa_y = 5000$ N/m; 3) $A_x, A_y = 30$ N; 4) $\upsilon = 0.3$ (where maximum joint velocity $\upsilon_{\max} = 1$); and 5) $f_n = 10$ N. For the hull, we used the following values for the robot motion parameters: 1) $f_x = 6$ Hz; 2) $\kappa_x, \kappa_y = 5000$ N/m; 3) $A_x, A_y = 35$ N; 4) $\upsilon = 0.1$ (where maximum joint velocity $\upsilon_{\max}$ =1); and 5) $f_n = 20$ N. These parameters were found to be optimal by using the method described in Section VIII. In the learning method, we used $C_p^{th} = 0.60$ and $P^{th} = 0.10$ for the bowl. For the ship hull model, we used $C_p^{th} = 0.90$ and $P^{th} = 0.30$. For the ship hull, we considered a two pass cleaning, the first pass for dry cleaning by scrubbing; the second pass with a wet sponge to remove the dust particles.

The candidate setups were generated by uniformly sampling the 3-D configuration space $(x, y, \alpha$, where α is the angle

TABLE VII
RESULTS FROM PHYSICAL EXPERIMENTS

Part	Solution Size	Node Expansion	n_{rfc}	Avg. Setup Time(s)	Setup Planning Time(s)
Bowl	5	52	360	59.31	58
Hull	2	8	192	46.12	32

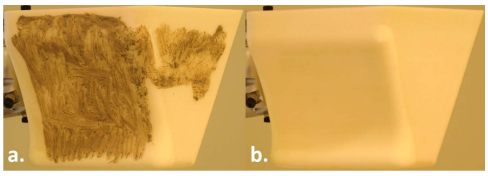

Fig. 12. 3-D printed ship hull model surface: (a) before cleaning and (b) after cleaning.

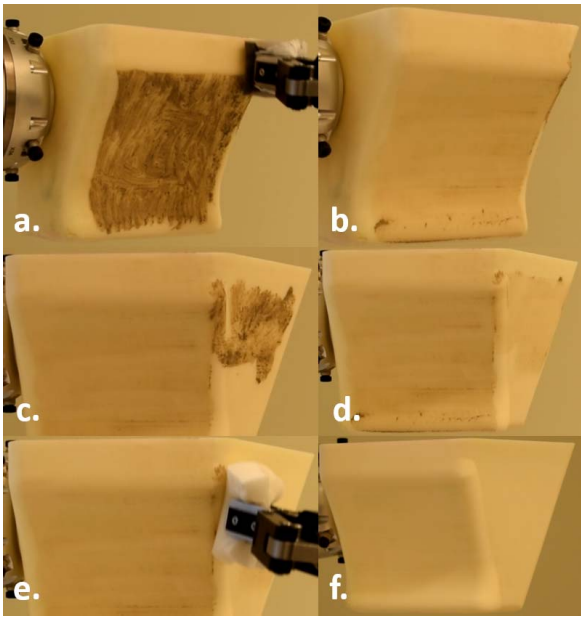

Fig. 13. Snapshots from a video showing execution of cleaning on the 3-D printed model of ship hull. This part has convex, concave, and flat regions on the surface. The setup planner generated a two-setup solution. (a) Ongoing cleaning on setup 1. (b) Setup 1 after cleaning by scrubbing. (c) Setup 2 before cleaning. (d) Setup 2 after cleaning by scrubbing. (e) Robot wiping off the dust from the surface. (f) Cleaned surface.

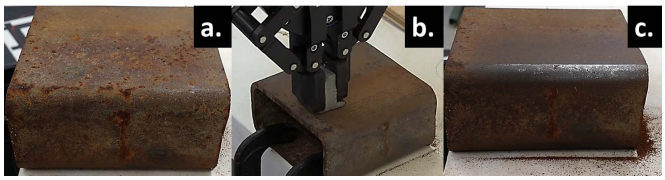

Fig. 14. Robotic cleaning of a rusty surface. (a) Before cleaning. (b) Ongoing cleaning pass. (c) After seven cleaning passes.

about z axis). We used CA_XY type gradient descent on top of hierarchical uniform sampling as they performed well in the synthetic tests. We used our setup planner described in Algorithm 2 to generate a solution. Table VII summarizes computational data of the setup planner from the physical experiments. For the bowl, it took three and two cleaning passes for setup 1 and all other setups, respectively. For the hull, it took a single cleaning pass for all the setups.

Figs. 10 and 12 illustrate the before-and-after cleaning states of the bowl and hull, respectively. Figs. 11 and 13 illustrate snapshots from a video of execution of the cleaning experiments. We also tested our system on cleaning rust on a metal block. The target region to be cleaned was user defined in this experiment. Fig. 14 illustrates the cleaning performance on a setup. The metal piece started to shine after seven cleaning passes. Our approaches for stain detection, cleaning performance evaluation, and estimation of the required number of cleaning passes are described in Section VII. The operation parameter selection method, described in Section VIII, ensures the selection of the right set of operation parameters to achieve the desired cleaning performance for each cleaning pass. In this paper, we referred to the required number of cleaning passes as N_c. In our current implementation N_c is user defined. If the estimation of N_c is correct, then there will be no residue stain. If the estimation of N_c is lower than what is needed, then there will be stain residue. The perception system tells the robots to stop cleaning when there is no more stain residue, even if N_c was over-estimated initially. As described in Section VII, a supervised learning method can be applied to estimate N_c accurately from the experimental data. However, in our current implementation we evaluate the cleaning performance on the entire part once the robot completes cleaning the part. If there is an error in estimating N_c, then we correct for it and run our setup planner again for the stain residues and iterate the cleaning process until there is no more stain left.

X. Discussion

We have conducted physical experiments on objects with constant curvatures. Therefore the operation parameter selection method produced results for fixed curvature. The right set of parameters may vary with curvature for the same surface and stain profile. Our parameter optimization method is general enough to add an arbitrary number of input parameters. Curvature of the surface can be added as an input parameter in the algorithm and then it will be able to determine the optimal set of parameters for different curvatures on the same surface.

In our experiments, we have considered a fixed pose of the robot with respect to the global frame of reference. The pose of the object is different with respect to the global frame for each setup. Therefore, the pose of the object is different with respect to the robot for each setup. The setup change time is the time taken to change the object's pose from one setup to another. Our setup planner is general enough to be used for objects too large compared to the dimensions of the robot. In that case the object can be stationary with respect to the global frame of reference. The setups will then correspond to different poses of the robot's base with respect to the global frame. The setup time will be the time to change pose of the robot's base with respect to the fixed object frame or global frame.

XI. Conclusion

We presented algorithms to automate robotic cleaning of curved surfaces with hard stains. The approach is applicable to tasks like polishing and paint stripping. Involvement of multiple repositioning and reorienting makes the method well suited for practical cleaning tasks. We presented heuristics that enable the algorithm to provide solutions in real time. Semi-supervised learning of optimal cleaning task parameters makes the method robust against change in surface and stain profile. We demonstrated experimental results where the robot removed stains that are tedious and difficult for humans to remove. In our physical experiment with a bowl, we considered fixture free operation. In our current implementation, a human operator changes the setups for fixture free operation. We plan to develop planning algorithms for a combination of prehensile and nonprehensile manipulations to automate the setup change in our future work.

References

[1] A. M. Kabir *et al.*, "Planning algorithms for multi-setup multi-pass robotic cleaning with oscillatory moving tools," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2016, pp. 751–757.

[2] R. Bormann, F. Weisshardt, G. Arbeiter, and J. Fischer, "Autonomous dirt detection for cleaning in office environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2013, pp. 1260–1267.

[3] F. Sato, T. Nishii, J. Takahashi, Y. Yoshida, M. Mitsuhashi, and D. Nenchev, "Experimental evaluation of a trajectory/force tracking controller for a humanoid robot cleaning a vertical surface," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2011, pp. 3179–3184.

[4] C.-H. King, T. L. Chen, A. Jain, and C. C. Kemp, "Towards an assistive robot that autonomously performs bed baths for patient hygiene," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, vol. 1, Oct. 2010, pp. 319–324.

[5] F. Nagata, T. Hase, Z. Haga, M. Omoto, and K. Watanabe, "CAD/CAM-based position/force controller for a mold polishing robot," *Mechatronics*, vol. 17, nos. 4–5, pp. 207–216, May/Jun. 2007.

[6] D. J. Buckmaster, W. S. Newman, and S. D. Somes, "Compliant motion control for robust robotic surface finishing," in *Proc. 7th World Congr. Intell. Control Autom. (WCICA)*, Jun. 2008, pp. 559–564.

[7] J. Hess, G. D. Tipaldi, and W. Burgard, "Null space optimization for effective coverage of 3D surfaces using redundant manipulators," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2012, pp. 1923–1928.

[8] C. Eppner, J. Sturm, M. Bennewitz, C. Stachniss, and W. Burgard, "Imitation learning with generalized task descriptions," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2009, pp. 3968–3974.

[9] A. Gams, M. Do, A. Ude, T. Asfour, and R. Dillmann, "On-line periodic movement and force-profile learning for adaptation to new surfaces," in *Proc. 10th IEEE-RAS Int. Conf. Humanoid Robots (Humanoids)*, Dec. 2010, pp. 560–565.

[10] B. Nemec and A. Ude, "Action sequencing using dynamic movement primitives," *Robotica*, vol. 30, pp. 837–846, Sep. 2012.
[11] D. Martínez, G. Alenyá, and C. Torras, "Planning robot manipulation to clean planar surfaces," *Eng. Appl. Artif. Intell.*, vol. 39, pp. 23–32, Mar. 2015.
[12] A. M. Kabir, J. D. Langsfeld, C. Zhuang, K. N. Kaipa, and S. K. Gupta, "Automated learning of operation parameters for robotic cleaning by mechanical scrubbing," in *Proc. ASME 11th Int. Manuf. Sci. Eng. Conf.*, Jun. 2016, p. V002T04A001.
[13] J. D. Langsfeld, A. M. Kabir, K. N. Kaipa, and S. K. Gupta, "Robotic bimanual cleaning of deformable objects with online learning of part and tool models," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2016, pp. 626–632.
[14] J. D. Langsfeld, A. M. Kabir, K. N. Kaipa, and S. K. Gupta, "Online learning of part deformation models in robotic cleaning of compliant objects," in *Proc. ASME 11th Int. Manuf. Sci. Eng. Conf.*, Jun. 2016, p. V002T04A003.
[15] W. X. Ng, H. K. Chan, W. K. Teo, and I. M. Chen, "Programming a robot for conformance grinding of complex shapes by capturing the tacit knowledge of a skilled operator," *IEEE Trans. Autom. Sci. Eng.*, to be published, doi: 10.1109/TASE.2015.2474708.
[16] Y. Zhao, J. Zhao, L. Zhang, L. Qi, and Q. Tang, "Path planning for automatic robotic blade grinding," in *Proc. Int. Conf. Mechatronics Autom. (ICMA)*, Aug. 2009, pp. 1556–1560.
[17] Y. T. Wang and Y. J. Jan, "Path planning for robot-assisted grinding processes," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 1, May 2001, pp. 331–336.
[18] W. Sheng, H. Chen, N. Xi, and Y. Chen, "Tool path planning for compound surfaces in spray forming processes," *IEEE Trans. Autom. Sci. Eng.*, vol. 2, no. 3, pp. 240–249, Jul. 2005.
[19] D. C. Conner, A. Greenfield, P. N. Atkar, A. A. Rizzi, and H. Choset, "Paint deposition modeling for trajectory planning on automotive surfaces," *IEEE Trans. Autom. Sci. Eng.*, vol. 2, no. 4, pp. 381–392, Oct. 2005.
[20] K. P. Hawkins, C.-H. King, T. L. Chen, and C. C. Kemp, "Informing assistive robots with models of contact forces from able-bodied face wiping and shaving," in *Proc. IEEE RO-MAN*, Sep. 2012, pp. 251–258.
[21] Y. K. Liu and Y. M. Zhang, "Supervised learning of human welder behaviors for intelligent robotic welding," *IEEE Trans. Autom. Sci. Eng.*, to be published, doi: 10.1109/TASE.2015.2453351.
[22] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, 1972.
[23] G. Hermann, "Algorithms for real-time tool path generation," in *Geometric Modeling for CAD Applications*. Amsterdam, The Netherlands: North-Holland, May 1988, pp. 295–305.
[24] Y. D. Chen, J. Ni, and S. M. Wu, "Real-time CNC tool path generation for machining IGES surfaces," *J. Manuf. Sci. Eng.*, vol. 115, no. 4, pp. 480–486, 1993.
[25] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Proc. 3rd Int. Conf. 3-D Digit. Imag. Modeling*, Jan. 2001, pp. 145–152.

Ariyan M. Kabir is currently pursuing the Ph.D. degree with the Department of Aerospace and Mechanical Engineering, University of Southern California, Los Angeles, CA, USA.

His current research interests include planning, learning, control, and perception algorithms for bi-manual manipulation to automate non-repetitive tasks.

Krishnanand N. Kaipa is currently an Assistant Professor with the Department of Mechanical and Aerospace Engineering and the Director of Collaborative Robotics and Adaptive Machines Laboratory with Old Dominion University, Norfolk, VA, USA. His current research interests include biologically inspired robotics, collaborative robotics, cognitive robotics, and swarm intelligence.

Jeremy A. Marvel (M'10) is currently a Computer Scientist and Project Leader with the Intelligent Systems Division of the U.S. National Institute of Standards and Technology, Gaithersburg, MD, USA. His current research interests include intelligent and adaptive solutions for robot applications, with particular attention paid to human-robot and robot-robot collaborations, industrial robot safety, machine learning, perception, and automated parameter optimization.

Satyandra K. Gupta is Smith International Professor with the Aerospace and Mechanical Engineering Department and the Director of the Center for Advanced Manufacturing with the University of Southern California, Los Angeles, CA, USA. His current research interests include computer aided design, manufacturing automation, and robotics.