SSK: Robotic Pen-Art System for Large, Nonplanar Canvas

Daeun Song, *Member, IEEE*, Jiyoon Park, and Young J. Kim, *Senior Member, IEEE*

***Abstract*—We present a semiautonomous robotic pen-drawing system, called SSK, that is capable of creating pen art on a large nonplanar surface. Our robotic system relies on a seven-degree-of-freedom impedance-controlled manipulator with a three-degree-of-freedom holonomic mobile platform. We use a vector-graphics engine to take an artist's pen drawing as input, and we generate Bézier spline curves to be drawn on the given target drawing canvas. Then, our system finds a set of minimal, discrete configurations for the mobile platform to cover the entire canvas surface while considering the reachability of the manipulator. The drawing is split into multiple subdrawings according to the found configurations. Our system replicates the spline drawing on the target surface using impedance control, which enables us to compensate for the uncertainty and incompleteness inherent to canvas-surface representations and various robotic and sensor noises. We demonstrate that our system can create visually pleasing and complicated pen art on large, nonplanar surfaces.**

***Index Terms*—Computational geometry, mobile manipulation, motion and path planning, robotic art.**

I. Introduction

SINCE the Renaissance, artists have incorporated machines and new technologies into artistic installations and performances to go beyond conventional art forms. Many contemporary artists are radically expanding the possibilities of creative expression using new media and mechanisms, and robots are increasingly becoming one of the mechanisms [1], [2], [3], [4]. In accordance with the recent impressive development of robotics technology, "robotic art" is becoming popular both for artists and roboticists.

The robotic art community has presented many fully-autonomous or semiautonomous robotic drawing systems in the past [5]. However, creating an autonomous robotic drawing system is just as difficult as creating any autonomous robotic system because it requires robust components of control, sensing, planning, and human–robot interfacing. What makes the matter more challenging is that the artistic nature of robotic art requires the interdisciplinary participation of, for instance, art and computer graphics technology, in addition to general robotic technology.

In order for the robots to reproduce an artistic workpiece, robots not only need to determine a sequence of drawing strokes, but also need to plan the proper motions to realize the sequence. Nonphotorealistic rendering (NPR) techniques in computer graphics may provide some clues as to how to convert digital images into painterly or sketchy strokes that can be executed by machines [6], [7]. In addition, vector-graphics images [8], unlike raster images, naturally provide drawing sequences or paths for the robot. After the drawing sequences and paths are decided, the robot can perform drawing by following the path.

The robotic pen-art system imposes many new challenges, unlike robotic painting. For instance, robotic pen art requires that the robot maintain constant contact with the target drawing surface to draw the path; this is known to be a difficult task, especially in the presence of uncertainty [9]. Impedance control, an approach for controlling the relationship between the force and the position, has become a popular choice for contact-intensive tasks [10], because it can deal with position uncertainties and adjust the robot's compliance according to the external force [11].

Meanwhile, most of the existing artistic robot-drawing systems are limited to the planar canvas. Drawing on a nonplanar canvas requires mapping the 2-D drawing path to a 3-D path on the target canvas surface without introducing visual artifacts. Thus, a reliable and distortion-free method is required to map between the 2-D drawing-path space and the 3-D canvas space. This 2-D–3-D mapping problem has been widely studied in computer graphics, particularly in the texture mapping domain [12]. Least squares conformable mapping (LSCM) [13] is a well-known distortion-free mapping method, particularly for preserving angle distortion. Finally, the reachability of a robotic manipulator can limit the size of the drawing canvas. The use of a mobile manipulator is a natural choice to address this problem. However, adopting such a platform brings the additional challenge of navigation planning to the system. More precisely, the system will need proper "coverage planning" for the mobile manipulator to be able to cover the large drawing canvas. Finding the path for a mobile manipulator with a high degree of freedom (DoF) to cover a 3-D surface can be computationally very expensive (e.g., NP-hard [14]).

Manuscript received 18 July 2022; revised 13 January 2023; accepted 11 April 2023. Date of publication 2 May 2023; date of current version 8 August 2023. This work was supported in part by the ITRC/IITP program under Grant IITP-2023-2020-0-01460, and in part by the National Research Foundation (NRF) in South Korea under Grant 2021R1A4A1032582 and Grant 2022R1A2B5B03001385. This paper was recommended for publication by Associate Editor M. Walter and Editor W. Burgard upon evaluation of the reviewers' comments. *(Corresponding author: Young J. Kim.)*

The authors are with the Department of Computer Engineering, Ewha Womans University, Seoul 03760, South Korea (e-mail: daeunsong@ewhain.net; jiyoonpark13@ewhain.net; kimy@ewha.ac.kr).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TRO.2023.3268585.

Digital Object Identifier 10.1109/TRO.2023.3268585

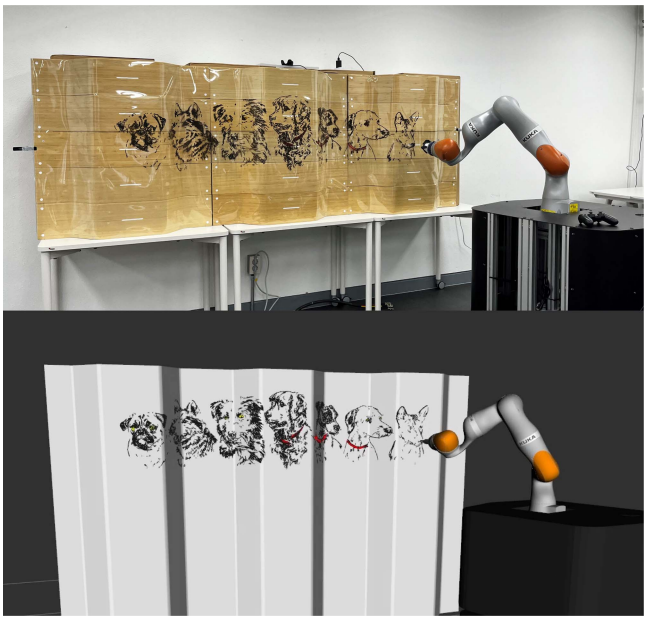


Fig. 1. Presented robotic pen-art system drawing artwork on a large, nonplanar canvas in the real-world (top) and in simulation (bottom).

A. Main Results

In this article, we present a robotic pen-art system, called SSK, that is capable of creating artistic pen art on a large, nonplanar surface (Fig. 1). We process the user's 2-D drawing through our vector-graphics engine to generate the input, which will later be translated into a drawing path. In order to map the 2-D drawing path onto the 3-D surface with minimal distortion, we apply LSCM. We also introduce a new *coverage planning* method for the mobile platform that enables our system to draw on a canvas whose size is larger than the manipulator's reachability when its base frame is fixed. In order to solve the large canvas drawing problem, we reformulate it as a combinatorial optimization problem that finds the minimum number of mobile base configurations that can cover the entire target drawing path, and we solve it using a greedy algorithm. Then, for each of the base configurations, we perform the robotic drawing task on a subset of the target canvas. We use impedance control to draw on a nonplanar canvas, which can also compensate for various error sources, such as the geometric uncertainties of the nonplanar canvas surface and the robot's localization error. We carry out a variety of drawing experiments to show that our system can create visually pleasing and complicated pen art on large, nonplanar surfaces. We also demonstrate that our robotic drawing system is robust to different sources of modeling, sensing, and localization errors.

The rest of this article is organized as follows. We survey works relevant to the robotic drawing system in Section II. Then, we outline an overview of the system architecture in Section III. We propose our drawing input generation method using a vector-graphics engine in Section IV. The details of mapping the produced 2-D drawing path onto the target 3-D surface are elaborated in Section V. A detailed description of planning the large canvas drawing task is presented in Section VI, and robotic curve rendering is presented in Section VII. We show our implementation results and discuss them in Section VIII. Finally, Section IX concludes the article.

Preliminary versions of this article have appeared in such conferences as in [15] and [16]. Based on these versions, we extend our previous robotic drawing system to cope with a large canvas using a mobile manipulator in this article. We efficiently solve the coverage problem using precomputed manipulator coverage maps and a greedy algorithm. We also provide new experimental results for robotic drawing on large, nonplanar surfaces using our improved system; we discuss the results, including error analysis, more extensively compared than we did in the preliminary versions.

II. Previous Work

A. Robotic Curve Rendering

The early work on creating drawing machines can be attributed to artistic work by Jean Tinguely and Harold Cohens Aaron [17]. In the computer graphics and robotics community, early attempts to create drawing robots were largely based on a plotter-type, special-purposed machine. More recently, research efforts have been applied to using a high-DoF general robot or manipulator for robotic drawing that can span a wide spectrum of artistic expressions.

There was an attempt to follow human-characteristic styles to draw human portraits using the HOAP-2 humanoid robot [2]. Paul the robot [4] is a robotic installation that creates observational portrait drawings that mimic an artist's stylistic signatures. The PumaPaint project [18] is a telerobotic painting robot that allows online users to draw paintings remotely using a PUMA robot. e-David [3] is a modified, industrial robot that can create a wide variety of painting styles from an image input. It relies on visual feedback to generate NPR-type painterly results, and has been extended to generate sets of shapes by analyzing various artist styles using region-based approach [19]. Busker Robot [20], [21] is a robotic painting system that paints watercolor artworks using NPR techniques, and it has been extended to create artworks using the palette knife techniques [22]. The authors also proposed a human–robot interaction architecture to draw an artistic drawing using an eye-tracking interface [23]. In drozBot [24], they formulated the drawing problem as a coverage problem and solved it using an ergodic control algorithm. This approach is similar to ours, only that we consider the larger coverage problem using a mobile manipulator.

Recently, with the development of machine learning technology, reinforcement learning that enables the painting agent to learn where to place the brush strokes has been studied [25]. Cloudpainter [26] took advantage of the machine learning style transfer method to apply the visual style of a source image to a target image. There is also a robotic pencil sketching system that uses neural style transfer technology to extract the content and style features and generate a new artwork [27]. The authors used a closed-loop force control to compensate for the pencil wear.

Although most of the research focuses on creating a drawing on a limited space of the flat canvas, there was recently an

attempt to draw a pen drawing on a 3-D surface using closed-loop planning and vision-force feedback [28]. However, none of the existing works dealt with creating a pen drawing on a large, nonplanar surface using a mobile manipulator.

B. Vector Graphics

For the robotic drawing system, vector graphics are more suitable than raster graphics, as vector graphics can generate continuous and smooth pen strokes that can be mapped nicely to smooth robotic motions. Vector graphics typically fill pixels inside implicitly-defined curves of a certain width using CPU-based scanline methods [29], [30], [31]. Because vector-graphics techniques need to render implicit curves in every frame, the performance of CPU-based rendering methods is slow on the dense screen resolution used by modern display devices.

Loop and Blinn suggested a GPU-based fast resolution-independent rendering method that could render paths and bounded regions [32]. Kilgard and Bolz introduced a fast GPU-based, two-step approach, namely the stencil-and-cover approach [33]. The stencil step determines the stroke path's fill coverage, and the cover step fills the area determined by the stencil step. There exists no vector-graphics method that can adequately reproduce human drawing consisting of free-form lines and curves, even though smooth curve rendering or retrieving methods, such as in [32], [34], [35], and [36], may handle human drawings to some degree.

C. Distortion-Free Surface Parameterization

The automatic parameterization of a nonparametric surface, like polygonal or point-set surfaces, is a nontrivial problem; it has been extensively studied in geometric modeling, and processing [37], [38]. Typical applications of parameterization include texture mapping in computer graphics and mesh editing and remeshing in geometric processing. In particular, the former application is closely related to our problem, where a curved drawing in 2-D should be faithfully reproduced on a curved or bumpy surface in 3-D.

For a polygonal surface with a disk-like topology and a convex boundary, barycentric mapping [39] is the most commonly used method in the literature. However, this method can induce serious distortion in parameterization if the boundary is highly nonconvex. To cope with this problem, conformal mapping based on complex analysis has been introduced. At a high level, conformal mapping can be classified into analytic and geometric methods [38]. The former is relatively easy to implement using energy minimization [13] but can suffer from unbalanced distortion if the underlying surface has a high Gaussian curvature, whereas the latter can resolve this issue [40].

D. Mobile Base Placement

In order to extend our robotic drawing system to draw on larger spaces, we need to incorporate the problem of finding a proper robot base position. Such a problem has been studied in many robotic fields that require accomplishing the task in a larger area than the robot's reachability, e.g., spray-painting robots, agriculture robots, and inspection robots. In [41], the authors presented a two-step search algorithm for determining the nearly optimal base position for a mobile painting manipulator. In [42], they proposed a method to find the optimum robot configurations while minimizing the number of mobile base movements and changes in manipulator configurations using the genetic algorithm. In [43], the authors considered the shape of the local workspace of a planar cable-suspended robot and evaluate the efficiency [44].

Although the prior methods can be applied in our system, since optimality is not our main concern, we approximate the problem using a simple geometric set-cover problem to solve it within a few seconds.

E. Impedance-Controlled Robot

Impedance control for controlling the interaction between a manipulator and the surrounding environment was first suggested in [45]. Since then, robot interaction techniques that use impedance control have been explored in the robotics community [46], [47]. The popularity of the use of impedance control is partly due to the demand for robotic systems with the ability to interact with uncertain environments in practical applications. Thus, many collaborative robots that are designed with impedance control have been introduced in both industry and academia, for instance, LBR iiwa from KUKA Robotics, Baxter and Sawyer from Rethink Robotics, and Justin from DLR. These robots are able to perform classically challenging robotic manipulation such as peg-in-hole assembly. Impedance control can also be used for unscrewing a jar lid, as done by the humanoid robot Justin [48], as well as bimanual tasks [49]. Impedance control in drawing was first introduced in our previous work [15], and it was used for compensating the uncertainty imposed by sensor noise and numerical noise in surface estimation.

III. System Overview

In this section, we present an overview of our system and the notation that is used in the article. As illustrated in Fig. 2, we present a system overview of our robotic drawing system and how to render a drawing on a nonplanar surface using robotic hardware. Our robotic hardware, consisting of a robotic manipulator placed on top of a mobile platform, is shown in Fig. 1. For convenience, we refer to a set of two robots as a robot. When referring to a specific robot, it is called either a manipulator or a mobile platform/base. Table I defines all the notations introduced in the article.

The input drawing of our system consists of a sequence of points, $\mathcal{D} \subset \mathbb{R}^2$, that define quadratic Bézier curve patches. In Section IV, we introduce a method for generating a drawing path that is drawn by artists using tablet-like pen interfaces. In order to create this 2-D drawing on a nonplanar 3-D surface with the least amount of distortion, we present a distortion-free conformal mapping method in Section V. Given the target surface data $\mathcal{S} \subset \mathbb{R}^3$ in a point cloud or mesh format, we estimate the normal vector for each point in $\mathcal{S}$. The normal vector is used for conformal mapping as well as for deciding the orientation

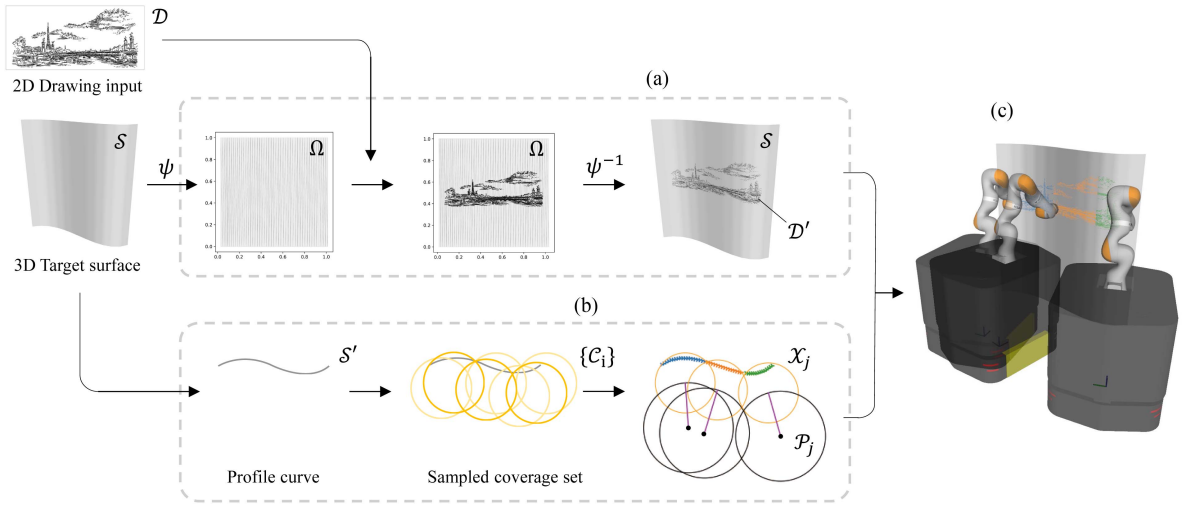


Fig. 2. System overview. Given (or after obtaining) the 2-D drawing input $\mathcal{D}$ and the 3-D target drawing surface data $\mathcal{S}$, (a) drawing mapping, the drawing is mapped onto the surface using LSCM, and (b) surface coverage planning, the surface coverage path is planned from the solution to our simplified drawing coverage problem. (c) Robotic rendering, the drawing is rendered on a target 3-D surface using robotic hardware. (a) Drawing mapping. (b) Surface coverage planning. (c) Robotic rendering.

TABLE I
TABLE OF NOTATION

Notation	Description
$\mathcal{D} \subset \mathbb{R}^2$	Point set defining 2D drawing
$\mathcal{S} \subset \mathbb{R}^3$	Point set defining 3D target surface
$\mathcal{S}' \subset \mathbb{R}^2$	Point set with only x and y values in $\mathcal{S}$
$\Omega \subset \mathbb{R}^2$	Surface point set in parametric domain for mapping
$\mathcal{D}' \subset \mathrm{SE}(3)$	Drawing pose set mapped onto target surface
$\mathcal{C}_i$	Coverage circle of the manipulator
$\mathcal{B}_i$	Bounding circle of the mobile base platform
$\mathcal{X}_j$	Covered surface point sets
$\mathcal{P}_j$	Target mobile platform poses

of the robot's end-effector. During the mapping process, the set of surface points is parameterized into a parametric domain, $\psi : \mathcal{S} \subset \mathbb{R}^3 \rightarrow \Omega \subset \mathbb{R}^2$. The original input drawing defined by $\mathcal{D}$ is made to fit within the domain of Ω, and its corresponding parameters of the drawing points are found by inverse-mapping Ω to $\mathcal{S}$. We get the drawing to pose set $\mathcal{D}' \subset \mathrm{SE}(3)$ mapped onto the target surface as a result. This conformal mapping process can be replaced by other mapping methods (e.g., projective texture mapping) for which the drawing distortion is tolerable.

As our robotic hardware is not limited to its drawing canvas size, our system is capable of drawing a large drawing on a large canvas space. Therefore, we need to plan a set of robot configurations that can reach the given drawing path. In order to solve this problem, we approximate and simplify the problem by introducing the term *coverage* in Section VI. Instead of finding the continuous robot configurations with the high degrees of freedom, we find a discrete set of robot coverage that covers the entire drawing path. The drawing is then split into multiple subdrawings according to the found set of coverage.

Then, the manipulator performs the robotic curve rendering for each subdrawing using impedance control. Impedance control enables our system to compensate for the possible errors induced by surface estimation and calibration. We repeat the manipulator's robotic curve rendering task and move the mobile base to the next pose until it finishes the entire drawing task.

IV. Drawing Input Generation

In this section, we introduce a method to generate a 2-D robotic drawing path for our system. We take an artist's pen drawing as input using a vector-graphics engine. Raster images, which consist of a discrete set of pixels, cannot be directly applied to robot motions, which require a set of continuous vectors. On the other hand, our vector-graphics engine generates a sequence of continuous vectors that can be mapped to the manipulator's continuous motion.

A. Vector Graphics Engine

The vector-graphics engine receives input points from pen-ready devices, such as tablet devices or mobile phones [Fig. 9(a)]. The stylus pen generates 2-D points along with the corresponding pen pressure. We convert these input points and pressures into vector-graphics output, preview them by rendering them, and provide them to a robotic manipulator to physically recreate the drawing on an arbitrary surface. The process can be represented by the following steps.

1) We filter out useless input points to reduce the size of the data, as illustrated in Fig. 3(b). Existing tablet-based pen-input devices typically produce sixty points per second, and these points may contain many useless and noisy points; e.g., many colinear points on a straight line. During filtering, we also need to consider the input pressure values to retain the variation in line thickness. To do this, we apply a median filter with a narrow range, say five points per one filtering step, to filter out the redundant points along a straight line. In this step, many colinear points will be eliminated. Then, we apply the bilateral filter to the filtered result. Due to the nature of the bilateral filtering method,

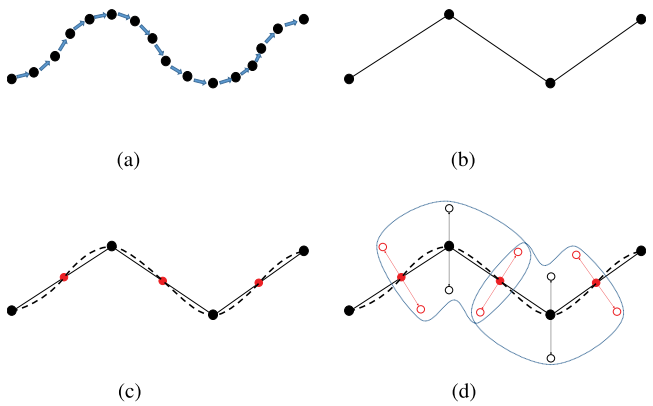


Fig. 3. Four steps for generating triangle polygon from input points for GPU-based vector-graphics rendering. (a) Input points. (b) Filtered points. (c) Midpoints generation (red). (d) Offset points generation for pen thickness.

points located near the crest of successive input points survive as shown in Fig. 3(b).

2) We calculate the midpoints of all successive input points.
3) We choose an input point as well as its two adjacent midpoints, calculated from the first step, to constitute three control points to define a single Bézier curve. This construction yields the C^1 continuity of the entire spline curve.
4) To render a curve with varying thickness (or offset), determined by the pen pressure, we triangulate the bounded areas, as illustrated in Fig. 3(d).

The robotic curve drawing requires the results of step 3, which are a set of quadratic Bézier curves with C^1 continuity and pressures. Additionally, in order to match the input required by our robotic hardware to operate the spline block motion, we use de Casteljau's algorithm to find a set of drawing points $\mathcal{D} \subset \mathbb{R}^2$ that passes the curve. Optionally, to preview the curve rendering with the tablet device before sending the input to the robot, we use the triangles of step 4 and render them using resolution-independent curve rendering, such as that in [32].

Although we provide an interface to generate a drawing from the user input, any kind of vectorized 2-D strokes can be used for the robotic curve rendering by our system.

V. Drawing From 2-D to 3-D

In order to reproduce a 2-D drawing on a 3-D arbitrary surface using robotic motions, one needs to decide the depth for each drawing point $\mathcal{D}$. Given a target surface point set $\mathcal{S} \subset \mathbb{R}^3$, either as raw point clouds or as a computer-aided design (CAD) software-generated mesh, the surface can be *unfolded* into a 2-D space. The depth of each drawing point can be decided simply by finding the correspondence between the drawing points and the unfolded 2-D surface, and then packing the correspondence back into 3-D space. There exist many different parameterization methods to unfold the surface, including the simplest one, which is projection mapping. To reproduce the drawing without severe distortion, one can employ conformal mapping. In other words, we can preserve the quality of the original 2-D drawing during mapping from a 2-D digital vector drawing to a robotic drawing

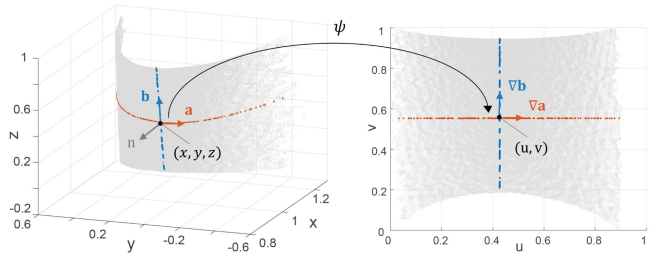


Fig. 4. LSCM. 3-D target surface data $\mathcal{S}$ in point cloud (left) and its parameterized representation Ω (right).

on a 3-D surface by minimizing the angle distortion of the local geometry. As a result, we obtain a distortion-free drawing result regardless of whether the drawing is executed on a large, small, stretched, or shrunken surface.

A. Conformal Mapping

Before explaining our idea of conformal mapping, we first introduce some theoretical background about conformal mapping. Given two surfaces with similar topology, it is possible to compute a one-to-one correspondence between them [37]. The problem of computing such a mapping is referred to as surface parameterization.

Conformal mapping is one of the surface parameterization techniques that preserves both angles and shapes. Let a continuous surface $\mathcal{S} \subset \mathbb{R}^3$ be parameterized into the parametric domain $\Omega \subset \mathbb{R}^2$. As illustrated in Fig. 4, a function ψ that is a mapping from 3-D surface S to the Ω domain in 2-D is said to be conformal if, for each $(u, v) \in \Omega$, the tangent vectors along the horizontal and vertical lines (the red and blue lines in Fig. 4), which form a regular grid, are orthogonal on S and have the same norm [38]

$$\mathbf{a} = \mathbf{n} \times \mathbf{b} \tag{1}$$

where $\mathbf{a}$ and $\mathbf{b}$ denote the tangent vectors and $\mathbf{n}$ denotes the unit normal at $(x, y, z) \in \mathcal{S}$. In other words, a conformal mapping locally corresponds to a similarity transform. It transforms an elementary circle on the surface to an elementary circle in the (u, v) domain.

B. Surface Drawing With Minimal Distortion

To realize conformal mapping in our work, we adopt LSCM [13] to parameterize the target surface. Although LSCM is applicable to nondevelopable surfaces, we only consider surfaces that do not require cutting to be *unfolded* into a 2-D parametric domain. After we unfold the target surface into the 2-D parameter space $(u, v) \in \Omega$, we search for the proper parameter values of the 2-D drawing data in the parameter space and *refold* the surface into 3-D space. We choose to compute the gradients using a local orthonormal basis of triangles, which requires that the surface be reconstructed as a triangular mesh. Alternatively, one could use a meshless technique for the conformal mapping of a point cloud without surface reconstruction by using the Laplace-Beltrami operator [50]. Both results would yield a set of coordinates $(u_i, v_i) \in \Omega$ associated with each point in P

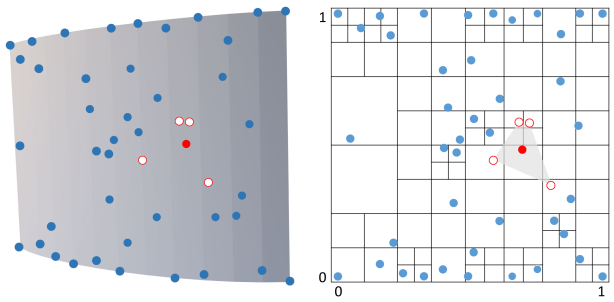


Fig. 5. Surface data in $\mathcal{S}$ (left) and Ω (right). The blue points represent the sparse surface points (point clouds or mesh), which have a corresponding point between $\mathcal{S}$ and Ω. The red solid point is a 2-D drawing point for which we want to estimate the depth. The four nearest neighbor points used for bilinear interpolation are highlighted in red outline.

that satisfies (1). In our implementation, we have chosen the meshing technique, as robust surface meshing implementations are readily available in [51], and as the meshing technique can be applied to both types of surface input, raw point clouds, and meshes.

After the surface parameterization ψ is complete, we need to decide the proper parameter coordinates for the drawing points set $\mathcal{D}$. Because our target surface data set $\mathcal{S}$ is an unorganized/sparse point set and does not necessarily form a uniform grid, in order to parameterize every drawing point to a corresponding (u, v) coordinate, we need to perform an estimation. Note that this is not a significant challenge for raster-based texture mapping, as one can obtain a clear idea of which rasterized point on the surface needs to be parameterized. On the other hand, in our case, there is no guarantee that all the drawing points in $\mathcal{D}$ can be mapped to the parametrized surface points in Ω. To solve this inverse mapping problem, we simply perform bilinear interpolation in the parametric domain to estimate the (u, v) coordinates for $\mathcal{D}$ as follows.

Given a desired drawing space $\mathcal{S}$ and a corresponding parameterized space Ω, surface data are stored in a quadtree data structure. Along with the desired drawing scale, we compute the parametric scale in Ω and fit the 2-D drawing to Ω. Then, for each drawing point $\mathbf{d}_i \in \mathcal{D}$, we search for the four nearest points in Ω that form a quadrilateral (Fig. 5). By performing bilinear interpolation on this quadrilateral, we can parameterize the $\mathbf{d}_i$ that is mapped to both the position $\mathbf{p}_i$ and the surface normal $\mathbf{n}_i$ on $\mathcal{S}$. Finally, a set of drawing points mapped to the target surface with the corresponding surface orientations is generated.

$$\mathcal{D}' = \{\langle \mathbf{p}_i, \mathbf{r}_i \rangle | \mathbf{p}_i \in \mathbb{R}^3, \mathbf{r}_i \in \mathrm{SO}(3)\}. \tag{2}$$

Thus far, we have assumed that every position and orientation is calculated in the *WORLD* reference frame (i.e., $\mathbf{p}_i^{WORLD}, \mathbf{r}_i^{WORLD}$), where we omit the indication for convenience.

VI. Large Canvas Drawing

Our system is not limited to the target drawing canvas size, as the robotic platform can move around freely. In particular, if the input drawing requires a large canvas, the manipulator will not be able to draw the whole drawing in one fixed place. Therefore, we need to find a set of robot configurations that can reach the given drawing path.

A. Problem Formulation

The objective of the large canvas problem is to find a path, ξ, for the robot that allows the manipulator to reach the given drawing path, $\mathcal{D}'$. This problem can be formally defined as follows.

Problem 1 (Drawing Coverage): Find a path ξ defined in the robot's configuration space[1] $\mathfrak{C}$ such that the end-effector of the robot following ξ is able to cover every drawing pose in the drawing path $\mathcal{D}'$ in (2). In other words

$$\mathcal{D}' \subset \bigcup_{\forall \mathbf{q} \in \xi} \mathcal{F}(\mathbf{q}) \tag{3}$$

where $\mathcal{F}$ is the forward kinematics map from $\mathfrak{C}$ to $\mathrm{SE}(3)$.

In general, finding ξ in Problem 1 with a high DoF robot is difficult. Adding a constraint or an objective would make the problem even more difficult. Thus, we reformulate the problem based on the notion of *path coverage*.

First, we define the robot's reachability and coverage as follows.

Definition 1 (Reachability): Given a robot $\mathcal{R}$ with its forward kinematics map $\mathcal{F}_\mathcal{R}$, the reachability $\mathcal{L}_\mathcal{R}(\mathbf{c})$ for a configuration $\mathbf{c} \in \mathrm{SE}(3)$ is defined as a mapping from $\mathrm{SE}(3)$ to $\{0, 1\}$ that indicates whether $\mathcal{F}_\mathcal{R}(\mathbf{q}) = \mathbf{c}$ for $\exists \mathbf{q} \in \mathfrak{C}$ (1) or not (0).

Definition 2 (Coverage): The coverage $\mathcal{C}$ for a robot $\mathcal{R}$ with reachability $\mathcal{L}_\mathcal{R}$ is defined as a set of points that are reachable by the end-effector of the robot. In other words

$$\mathcal{C} = \{\forall \mathbf{p} \in \mathbb{R}^3 \mid \exists \mathbf{q} \in \mathfrak{C}, \exists \mathbf{r} \in \mathrm{SO}(3), \mathcal{L}_{\mathcal{R}(\mathbf{q})}(\langle \mathbf{p}, \mathbf{r} \rangle) = 1\} \tag{4}$$

where $\mathcal{R}(\mathbf{q})$ represents the robot placed in a configuration $\mathbf{q}$.

Note that when $\mathbf{q}$ is bounded, $\mathcal{C}$ is compact. Then, instead of solving Problem 1 exactly, we first discretize the drawing path $\mathcal{D}'$ with a finite set of points $\mathcal{S}$, and we solve the following coverage problem to approximate Problem 1.

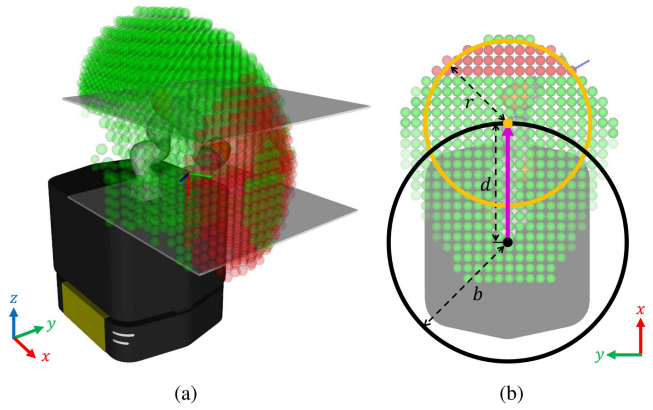
Problem 2 (Path Coverage): Given a finite point set $\mathcal{S}$ in 3-D that approximates the target drawing path $\mathcal{D}'$, find a minimal number of compact sets $\mathcal{C}_i$ such that their union includes $\mathcal{S}$

$$\min \left\{ n \mid \mathcal{S} \subset \bigcup_{i=1}^{n} \mathcal{C}_i \right\} \tag{5}$$

where each $\mathcal{C}_i$ denotes a subset of the robot's coverage $\mathcal{C}$ as defined by Definition 2.

While Problem 2 approximates the drawing coverage of Problem 1, it still requires complicated geometric and combinatorial optimization. Thus, we further simplify the problem by restricting each compact set $\mathcal{C}_i$ to be a simple geometric primitive in $\mathbb{R}^3$, such as a sphere, and precomputing it using the coverage of the robot. Furthermore, we use randomization to reduce the universe of $\{\mathcal{C}_i\}$ and rely on greedy-based combinatorial optimization to solve Problem 2. After a solution to Problem 3 is obtained, we

[1] In our case, $\mathfrak{C} = \mathrm{SE}(2) \times \mathbb{R}^7$.



(a) (b)

Fig. 6. Coverage map of the manipulator in 3-D with 0.05 m resolution (a), and its top-down view in the x-y plane (b). The spheres in the figures represent poses that are reachable by the manipulator. In particular, the red spheres are used to compute the $\mathcal{C}_i$ [the yellow circle in (b)] that are within the two parallel slabs [the gray planes in (a)] defined by the vertical height of the target drawing path. The black circle in (b) represents the mobile base's bounding circle $\mathcal{B}_i$, and the magenta arrow represents the heading of the mobile base.

use simple local planning to continuously switch the robot poses from one coverage to another coverage, which yields the entire robot path ξ.

B. Discretization and Reduction

In order to simplify the problem, we first introduce the notion of *coverage map*. The coverage map $\mathcal{M}$ for a robot $\mathcal{R}$ with reachability $\mathcal{L}_{\mathcal{R}}$ is defined as a set of points that are reachable by the end-effector of $\mathcal{R}$ when the base frame of $\mathcal{R}$ is fixed at $\mathbf{p}_{\text{base}} \in \text{SE}(2)$; i.e., the configuration space $\mathfrak{C}$ in (4) is restricted to $\mathbf{p}_{\text{base}} \times \mathbb{R}^7$. The coverage map is bounded and sometimes called the *reachability map* in the literature [52], [53]. In practice, the map is precomputed by discretizing both the configuration and the workspaces. In our case, we construct the coverage map as follows.

1) Discretize the Cartesian workspace in $\mathbb{R}^3$ with $\{\mathbf{p}_d\}$.
2) Discretize the configuration space in $\text{SO}(3)$ with $\{\mathbf{r}_d\}$.
3) For each $\mathbf{p}_d$, if $\mathcal{L}_{\mathcal{R}(\mathbf{p}_{\text{base}} \times \mathbb{R}^7)}(\langle \mathbf{p}_d, \mathbf{r_d} \rangle) = 1$ for all $\mathbf{r}_d$ using either forward kinematics or inverse kinematics, then add $\mathbf{p}_d$ to $\mathcal{M}$.

Fig. 6 shows the resulting coverage map $\mathcal{M}$ of our robot, where the green spheres correspond to each element in $\mathcal{M}$. Note that $\mathcal{M}$ is a discretized version of $\mathcal{C}_i$ when the base frame is fixed.

In general, the geometry of $\mathcal{C}_i$ may be highly nonconvex, and may contain holes due to kinematic limits [52]. Even though a simple geometric primitive like a sphere may not closely approximate the complicated geometry $\mathcal{C}_i$, we nonetheless opted for the sphere to approximate the geometry due to its simplicity, which will in turn ease the burden of reachability checking—the spherical approximation of $\mathcal{C}_i$ using $\mathcal{M}$ can quickly find reachable points on the target surface. We also reduce the coverage dimension in Problem 2 from 3-D to 2-D by assuming a certain geometric characteristic of the target surface. We assume that the geometry of our target canvas is a surface of extrusion (e.g., see Fig. 8)—the surface is *vertically* extruded along the z-axis from a 2-D planar curve (i.e., the profile curve) defined in the x-y plane. Thus, because the geometric characteristics of the target surface still pertain to the 2-D profile curve (or the 2-D projection of the surface), we can effectively project the spherical coverage of $\mathcal{C}_i$ in 3-D to a circle in 2-D. We also use another circle in 2-D to bound the mobile base, $\mathcal{B}_i$, which will be used in checking collisions with the canvas. The use of the bounding circle $\mathcal{B}_i$ not only allows the mobile base to remain collision-free, but also compensates for the area that is unreachable due to the spherical approximation of the geometry of $\mathcal{C}_i$. Similarly, the target drawing path $\mathcal{S}$ is also projected from 3-D to a 2-D point set $\mathcal{S}' \subset \mathbb{R}^2$.

Now, our path coverage problem in 2-D can be stated as follows,

Problem 3 (Circular Path Coverage): Given a finite point set $\mathcal{S}'$ in 2-D that approximates the drawing path, find a minimal number of coverage circles $\mathcal{C}_i$ such that their union covers $\mathcal{S}'$

$$\min \left\{ n \mid \mathcal{S}' \subset \bigcup_{i=1}^{n} \mathcal{C}_i,\ \mathcal{S}' \cap \bigcup_{i=1}^{n} \mathcal{B}_i = \emptyset \right\} \tag{6}$$

where $\mathcal{B}_i$ denotes the circle bounding the robot's mobile base relevant to $\mathcal{C}_i$.

1) Computing $\mathcal{C}_i$ and $\mathcal{B}_i$: A simple method is employed to decide the geometry of the circles that represent the coverage candidates $\mathcal{C}_i$ and the mobile base $\mathcal{B}_i$ in 2-D, as well as to determine the intercenter distance d between $\mathcal{C}_i$ and $\mathcal{B}_i$ [Fig. 6(b)]. First, from the precomputed geometric distribution of *cells* in $\mathcal{M}$, we observe the following results.

1) The overall discrete geometry of $\mathcal{M}$ is similar to a spherical shell, where the inner shell corresponds to the area that is unreachable by the manipulator.
2) We orient the mobile base in such a way that the drawing occurs only in the forward-facing direction of the base, i.e., the x-axis of the base frame. Thus, the main radial direction of the spherical shell $\mathcal{M}$ is aligned with the x-axis, and its radial extent is bounded.
3) We assume that the target drawing path (or the canvas) has a limited vertical height (along the z-axis). Thus, the spherical shell is truncated by the slabs defined by the vertical height.

The projected geometry of $\mathcal{M}$ onto the x-y plane looks like Fig. 6(b) and, based on the aforementioned observations, we bound only the first few consecutive rows (the red spheres in the figure) in the projected $\mathcal{M}$ using a circle (the yellow circle in the figure). These rows are contiguous so that they correspond to the internals of the spherical shell. Using the extrema of these rows, as well as the constraint that the circle's center is aligned with the x-axis, we determine the circle of $\mathcal{C}_i$. The circular bound $\mathcal{B}_i$ of the mobile base with some safety offset is also computed (the black circle in the figure); the intercenter distance d is also obtained from these two circles, $\mathcal{C}_i$ and $\mathcal{B}_i$. Finally, to better approximate the spherical-shell geometry of $\mathcal{M}$, in the next section, we use $\mathcal{C}_i - \mathcal{B}_i$ instead of using just $\mathcal{C}_i$, where "$-$" is the Boolean difference. In fact, (6) encodes this Boolean difference operation. Our method for finding $\mathcal{C}_i, \mathcal{B}_i$ is simple, yet conservative and effective in practice.

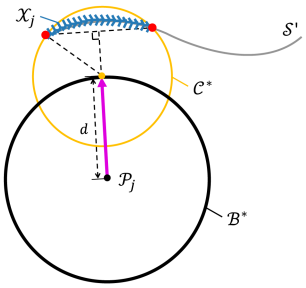


Fig. 7. Two extreme points (red dots) of $\mathcal{S}'$ inside the coverage circle $\mathcal{C}^*$ are identified. $\mathbf{p}_{\text{base}}$ is displaced from the center of $\mathcal{C}^*$ by d, the precomputed distance between the centers of two circles, and the orientation of $\mathbf{p}_{\text{base}}$ is normal to the line connecting the extreme points.

Algorithm 1: Greedy Algorithm for Set Cover.

Input : $\mathcal{S}'$: point sets in $\mathbb{R}^2$ for the target surface
Output: $\mathcal{T} = \{\langle \mathcal{P}_j, \mathcal{X}_j \rangle\}$: tuples of base poses in $\mathrm{SE}(2)$ and covered point sets in $\mathbb{R}^2$

1 $\mathcal{T} \leftarrow \emptyset$;
2 A set of densely-sampled candidate circles $\{\mathcal{C}_i\}$;
3 **while** $\bigcup \mathcal{X}_j \neq \mathcal{S}'$ **do**
4 $\quad \mathcal{C}^* = \underset{\mathcal{C}' \in \{\mathcal{C}_i\}}{\operatorname{argmax}} |(\mathcal{S}' - \bigcup \mathcal{X}_j) \cap \mathcal{C}'|$;
5 $\quad$ Find the base pose $\mathbf{p}_{base}$ and its bounding circle $\mathcal{B}^*$ corresponding to $\mathcal{C}^*$;
6 $\quad$ **if** $\mathcal{S}' \cap \mathcal{B}^* \neq \emptyset$ **then**
7 $\quad\quad X_j = \mathcal{C}^* \cap \mathcal{S}'$;
8 $\quad\quad \mathcal{P}_j = \mathbf{p}_{base}$;
9 $\quad\quad$ Add $\mathcal{T}_j = \langle \mathcal{P}_j, \mathcal{X}_j \rangle$ to $\mathcal{T}$;
10 $\quad$ **end**
11 $\quad$ Remove $\mathcal{C}^*$ from $\{\mathcal{C}_i\}$;
12 **end**

C. Set-Cover Algorithm

Now we solve the combinatorial optimization of Problem 3. This problem is essentially a set-cover problem, which is known to be NP-hard [54]. In our case, we rely on a rather simple but effective approach based on randomization and greedy algorithms. Moreover, we find the coverage set that covers the entire target surface, not just the drawing path; i.e., $\mathcal{S}'$ in (6) approximates the target surface. Thus, this optimization is independent of the drawing input and is precomputed in our system. Algorithm 1 summarizes an overview of our algorithm.

First, we populate the universe of $\{\mathcal{C}_i\}$ by densely random-sampling the circles in $\mathbb{R}^2$ until the union of the sampled circles covers the target surface (line 2). From among these candidate circles, we find the circle $\mathcal{C}^*$ that includes the maximum number of points on the target surface that have not been covered by the already-chosen circles (line 4). For such a $\mathcal{C}^*$, the set of all the surface points that are inside the circle $\mathcal{C}^*$ is $\mathcal{X}^* = \mathcal{C}^* \cap \mathcal{S}'$. Then, we find the configuration $\mathbf{p}_{\text{base}}$ of the robot base and its bounding circle $\mathcal{B}^*$ that corresponds to $\mathcal{C}^*$ (line 5); $\mathbf{p}_{\text{base}}$ is calculated by moving in the perpendicular direction of the line connecting the two extreme points in $\mathcal{X}^*$ (Fig. 7). After verifying that the robot's base is collision-free from the target surface when the robot is placed at $\mathbf{p}_{\text{base}}$ (line 6), we add $\mathcal{X}^*$ to $\mathcal{X}$, the set of already-covered points. We repeat this until the union of $\mathcal{X}_j$ includes all the target surface points (line 3). Fig. 8 shows examples for three different target surfaces.

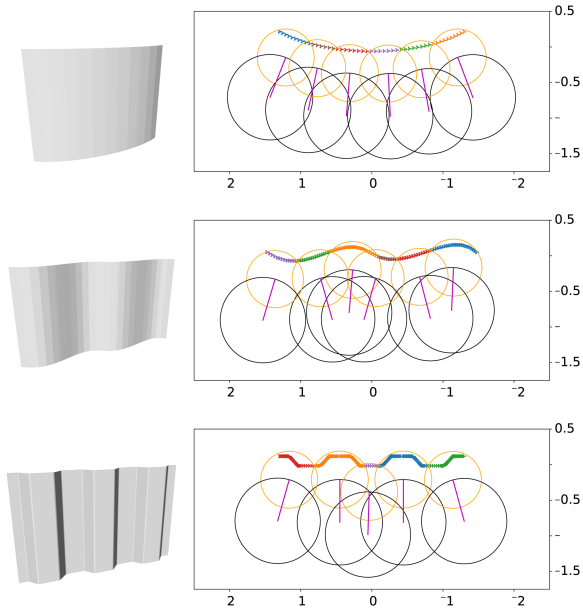


Fig. 8. Three canvas surface models in 3-D (circular, wave, curved) (left column), and the corresponding path coverage results projected in 2-D (right column). In the right images, different colored points on the surface represent the surface area covered by each candidate circle.

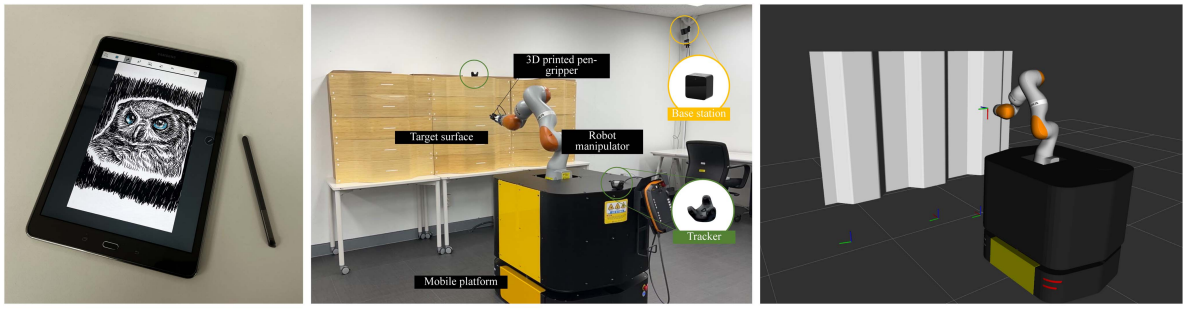
As a result of Algorithm 1, we obtain a set of tuples $\mathcal{T}_j = \langle \mathcal{P}_j, \mathcal{X}_j \rangle$ consisting of a mobile base pose $\mathcal{P}_j$ and the point set $\mathcal{X}_j$ covered by $\mathcal{P}_j$. We then partition the input drawing points $\mathcal{D}'$ into a subset of points $\mathcal{D}_j$ according to the coverage: i.e., $\mathcal{D}_j = \mathcal{X}_j \cap \mathcal{D}'$. $\mathcal{D}_j$ is fed into a manipulator, and the manipulator performs the robotic drawing task using impedance control (Section VII). The sequence of $\mathcal{P}_j$ values is spatially sorted and fed into the mobile base. Next, a linear interpolatory motion is employed to navigate from $\mathcal{P}_j$ to $\mathcal{P}_{j+1}$ successively. We repeat the alternation of drawing motion and mobile base navigation until we traverse the tuple set $\mathcal{T}$.

VII. Robotic Drawing

Finally, we carry out the drawing task using our robotic hardware. In this section, we explain how to carry out the drawing task while maintaining contact with the target surface using an impedance control manipulator.

A. Impedance-Controlled Drawing

Impedance-controlled robots interact with the environment by employing a mass–spring–damper-like system that actively controls the robots [10]. Such a system is well-suited to tasks in which contact forces should be kept small, whereas the accurate regulation of contact forces is not mandatory. Thus, impedance control serves nicely for pen drawing tasks. Using surface estimation and mapping, our drawing robot is now fully equipped with a set of drawing poses $\mathcal{D}'$ in 3-D that can be drawn on



(a) (b) (c)

Fig. 9. SSK system setup. (a) Vector-graphics engine. (b) Robotic drawing system. (c) Drawing simulation.

the target surface. However, to exert the proper, compliant force at the pen tip (the end-effector), as well as to compensate for possible estimation and sim-to-real errors, we adopt the Cartesian impedance control method. Impedance control is a hybrid, position-, and force-controlled method that was proposed for interaction between a robot and an unstructured environment [45]. By employing a mass–spring–damper-like system, impedance control allows a robot to react in a compliant manner to external obstacles. By considering a certain offset value (i.e., impedance) for each drawing point, Cartesian impedance control results in continuous contact with the surface in a Cartesian space during the entire drawing session.

The pen-tip attached to the robot manipulator is always perpendicular to the wall (2). We configured the impedance controller in such a way that the robot is compliant only in the pen-tip heading direction. Additionally, in order to exert an appropriate amount of compliant force at the pen-tip, a small deviation between the target position and the physical position of the pen-tip needs to be provided to the impedance controller.

Simply having the set of drawing points $\mathcal{D}' = \{\mathbf{d}'_i = (\mathbf{p}_i, \mathbf{r}_i) | \mathbf{p}_i \in \mathbb{R}^3, \mathbf{r}_i \in \mathrm{SO}(3)\}$ define the target drawing poses can result in a lack of sufficient pen pressure and may be very sensitive to surface estimation error. Therefore, the target position $\mathbf{p}_i$ of $\mathbf{d}'_i$ was modified to

$$\mathbf{p}'_i = \mathbf{p}_i - \gamma z \mathbf{n}_i \tag{7}$$

where γz is a user-defined gain value that controls the pen pressure, and $-\mathbf{n}_i$ is the direction opposite to surface normal, which is parallel to the z-axis of the pen frame. This deviation results in a compliant force $\mathbf{f}_z = k\gamma z$ along the z-axis of the pen frame where k is the spring stiffness. To be more efficient, instead of calculating new positions for every $\mathbf{d}'_i$, we attach a virtual frame V to the physical pen that is aligned with the pen-tip attached to the end of the pen, except that V is slightly offset by γz from the pen-tip frame along the z-direction, which has the same effect as moving the pen-tip frame to $\mathbf{p}'_i$.

As a result, the pen-type end-effector traces out the position of spline curves while maintaining contact with the surface, exerting an almost uniform level of compliant forces. Experimentally, we have chosen the target pen contact force as 2.5 N for geometric error compensation. Our system then repeats to draw and move to the next coverage using simple local path planning.

VIII. Experimental Results

In this section, we describe the implementation details and demonstrate the drawing results of our system. To provide a drawing result, we use two different types of drawing inputs: 1) the ones generated using our vector-graphics engine (Racoon, Kangaroo, Bear, and Owl); and 2) inputs downloaded from Adobe Stock[2] in Scalable Vector Graphics (SVG) format (Dogs, Farm, and Paris). We tested our drawing system on two types of nonplanar canvases: 1) a circular column wall captured by an RGB-D camera and reconstructed as a point cloud (Figs. 10 and 11); and 2) a wavy wooden wall with 3-D model data (Fig. 12). We placed a thin plastic cover on the physical canvas surface to preserve it.

A. Implementation Details

As shown in Fig. 9(b), our robotic drawing system consists of a KUKA LBR iiwa 7 R800 manipulator, equipped with a 3-D printed pen holder, placed on top of the omnidirectional mobile platform Ridgeback from Clearpath Robotics. We use Python and C++ for the programming infrastructure, which runs on a laptop equipped with Intel i7-10 CPU and 16-GB RAM. We use Robot operating system (ROS) Melodic framework under the Ubuntu 18.04 LTS operating system to communicate with the robots (sensors). The manipulator runs under Sunrise OS with a real-time interface provided by the manufacturer. We use the Samsung Galaxy tablet PC to run vector graphics under the Android operating system and an Intel RealSense ZR300 RGB-D camera to reconstruct a 3-D point cloud from the target canvas space. We use HTC VIVE tracker 3.0 with OpenVR[3] to localize both the robot and the target drawing wall. For drawing simulation and fast prototyping, we use the Gazebo simulator as shown in Fig. 9(c), with Rviz to visualize the drawing results and MoveIt! [55] with TRAC-IK [56] inverse kinematics solver for computing the virtual robot trajectory following the Cartesian path in the simulated world. To utilize the physical robot, we use a default planner provided by KUKA Robotics to carry out the spline motions.

[2][Online]. Available: https://stock.adobe.com/
[3][Online]. Available: https://github.com/ValveSoftware/openvr

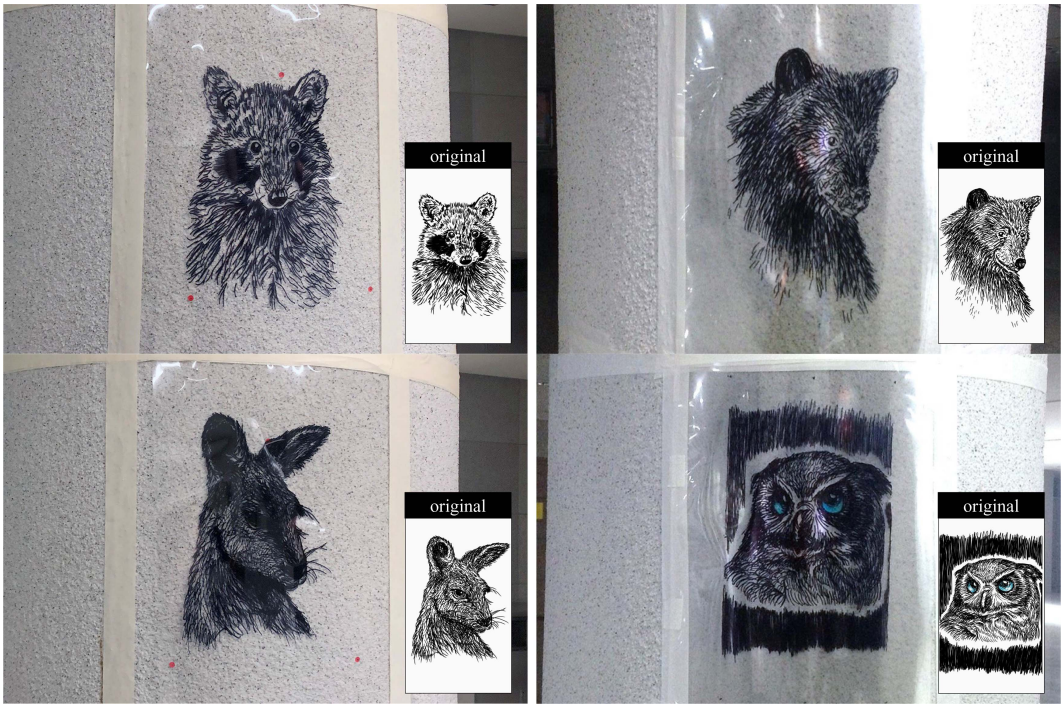


Fig. 10. Drawing results on a bumpy, circular column wall. Starting from the top left, Racoon, Bear, Owl, and Kangaroo, in the clockwise direction.

(a) (b) (c)

Fig. 11. Fractal curve drawing results on a bumpy, circular column wall. The black lines represent the robotic drawing results using conformal mapping, and the orange lines represent the results using projection mapping. Note that the orange lines are distorted relative to the black lines. (a) Grid with Hilbert space-filling curve. (b) Sierpiński arrowhead curve. (c) Koch snowflake curve.

B. Robotic Drawing Results

1) Artistic Drawing Results: Fig. 10 shows the artistic pen drawing results reproduced on a bumpy circular column wall using our system. The original drawings are drawn by the artist using our vector-graphics engine, and the target surface data is reconstructed as a point cloud using the RGB-D camera while the robot base remains fixed. The point cloud of the target surface consists of over 172 000 points. Our system is capable of drawing multiple color drawings by splitting the drawing tasks into different colored sets: e.g., the Owl drawing consists of two different colors, black and blue, in the eyes. However, because our 3-D printed pen holder has no tool-change mechanism, we manually change the pen when switching the drawing task as needed. The drawings use conformal mapping to map a 2-D drawing to 3-D, except for the Bear and Owl drawings (marked with a star in Table II), which use a simple orthographic projection (i.e., projection mapping).

2) Patterned Drawing Results: To highlight the effect of the conformal mapping, we show comparisons between the results that use conformal mapping and those that use the projection mapping by drawing fractal curves on a circular column wall (Fig. 11). Compared with the results for projection mapping, the conformal mapping reproduces the original drawing faithfully on the nonplanar surface. On the other hand, the projection mapping method does not preserve the original side length, and the length gradually increases as it moves away from the center of the projection. As can be seen from the grid pattern results shown in Fig. 11(a), the conformal mapping preserves the side lengths of the grid, but with the projection mapping, the length is increased by more than 20% in the worst case.

3) Large-Scale Drawing Results: Fig. 12 shows the artistic pen drawing results reproduced on a curved, wooden wall. The original drawings are SVG images that are translated to a set of 2-D strokes. The Dogs drawing consists of three different colors, yellow in some of the eyes, red on the leash, and black

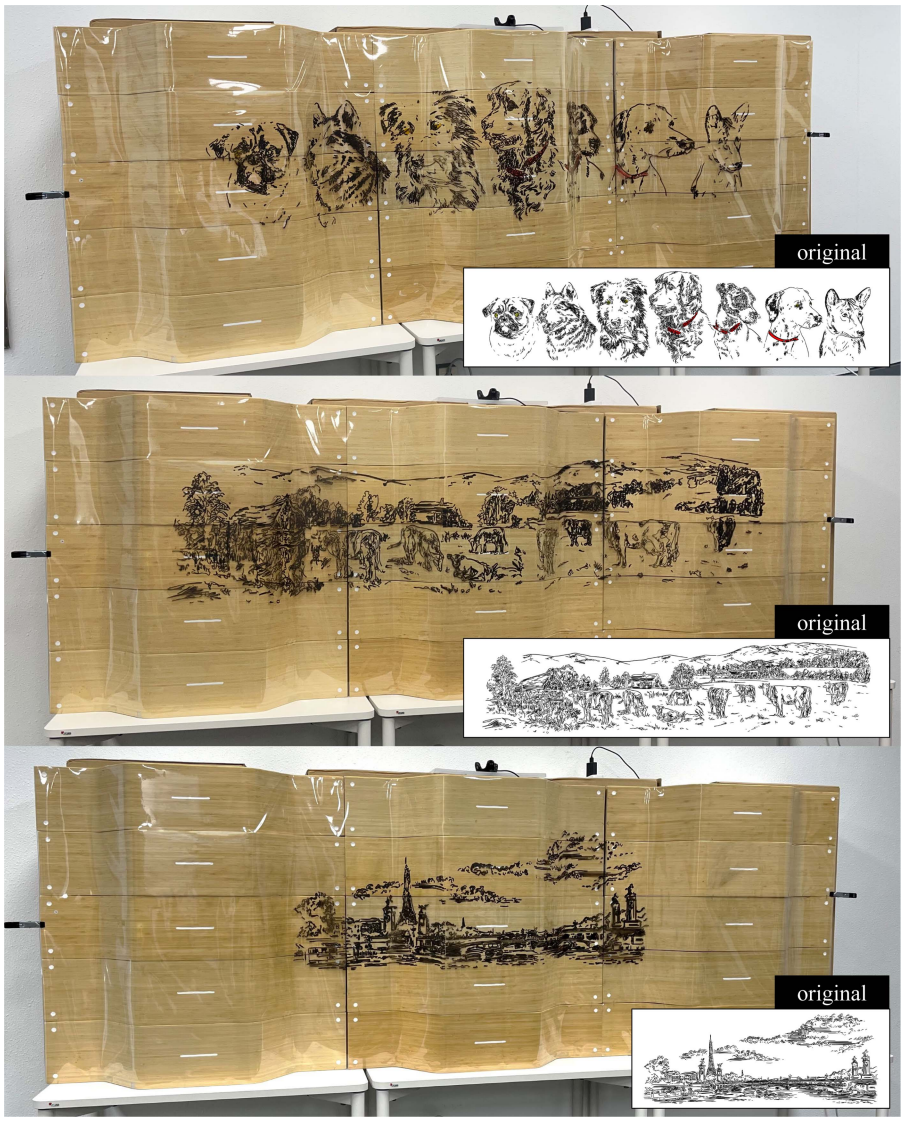


Fig. 12. Robotic drawing results: Dogs, farm, Paris, from top to bottom. Original vector-graphics image and the images drawn on a physical curved surface.

TABLE II
ROBOTIC DRAWING EXPERIMENTAL STATISTICS

Surface	**Circular column wall (Fig. 11)**				**Wooden curved surface (Fig. 10)**		
drawing	Racoon	Kangaroo	Bear*	Owl*	Dogs	Farm	Paris
Drawing size (mm)	252×491	252×491	252×491	252×491	2000×500	2000×500	1187×500
# of Coverage poses	1	1	1	1	5	5	3
# of Pen strokes	860	3,147	1,520	1,942	2,634	927	1,318
# of Drawing points	69,350	80,580	66,910	159,895	25,727	20,787	20,154
Drawing time (min.)	186	293	221	317	582	494	307

for the rest. The target surface data is modeled as a polygonal mesh in 3-D. There is a geometric discrepancy between the 3-D mesh model and the physical wall. We will discuss this error in Section VIII-D. Nonetheless, by using impedance control, our system successfully reproduces artistically-pleasing drawings on a large, nonplanar surface by compensating for the model discrepancy. The statistics for the experimental drawings shown in Figs. 10 and 12 are provided in Table II. The statistics for Fig. 11 are provided in Table III. They include the drawing size, the number of drawing strokes and points, and the drawing execution time. Robotic drawing by our system takes time because we executed our robot at a low speed, which was only 20% of the maximum joint velocity, for the safety of both the robot and any drawing collaborators (i.e., humans). Table II also includes the number of coverages (i.e., C_i) that are used to split the drawings. Table III also shows the mapping time to

TABLE III
CONFORMAL MAPPING AND DRAWING EXPERIMENTAL STATISTICS

Drawing	Grid	Arrowhead	Snowflake
Drawing Size (mm)	384×216	432×216	384×192
# of Pen strokes	95	12	12
# of Drawing points	6,930	2,360	4,128
Mapping time (s)	635	308	523
Drawing time (min.)	54	14	19

TABLE IV
SURFACE SET COVER EXPERIMENTAL STATISTICS

Surface	Circular	Wave	Curved
Dimension (cm^3)	$260 \times 30 \times 200$	$290 \times 20 \times 170$	$260 \times 15 \times 180$
# of vertices	98	194	752
# of faces	96	192	750
Set cover (s)	2.674	8.677	378.241

parameterize the surface, and then, to map the drawing using the method described in Section V.

C. Set-Cover Results

Fig. 8 shows three different nonplanar surfaces and the coverage-problem results using the greedy algorithm presented in Section VI-C. Table IV shows the geometric information for the surfaces and the computational performance of our algorithm for each case. The first row shows the physical dimension of the surface model. The second and the third rows show the number of vertices and the faces in the geometric model of the surface. The last row is the coverage-problem solution time using our greedy algorithm. Our greedy algorithm has a time complexity of $O(mn)$, where m and n represent the number of points comprising the target surface $\mathcal{S}'$ and the number of candidate circles $\mathcal{C}_i$, respectively, and takes only 0.001% of the entire robotic drawing process time.

D. Discussion Concerning Error

In our system, there exist different sources of error that are attributed to surface deviation, coverage approximation, and localization. Because these errors can affect the quality of the final drawing results, in the following, we address how we compensate for the errors in our system.

1) Surface Deviation: We reconstruct the canvas surface automatically using an RGB-D camera or manually using geometric modeling software. In both cases, there may exist some discrepancy between the physical and the geometric models. This type of surface deviation, up to 5 cm in our experiment, was successfully managed with the use of impedance control, as explained in Section VII.

To further validate the robustness of impedance-controlled drawing, we designed an experiment where we asked the robot to draw a 7×7 square grid, as shown in Fig. 13(a) on a hemisphere that was provided as a target surface geometry to the robot. However, the actual hemispheric surface was perturbed by 3.5% of the Hausdorff metric [57] relative to the size of the hemisphere and was 3-D printed as shown in Fig. 13(b). Without knowing this change, the robot attempts to draw the grid on the modified hemisphere. We measured how much the two geometric quantities of each grid deviated from the original hemisphere: 1) the 1-1 aspect ratio of the four sides; and 2) the right angles of the corners. In our experiments, the robot successfully drew the grid, and the relative deviations of the aspect ratio and the corner angles were measured as 3.0% and 4.4%, respectively. These deviations show a rate of changes similar to that of the 3.5% surface deviation, thanks to the impedance control scheme.

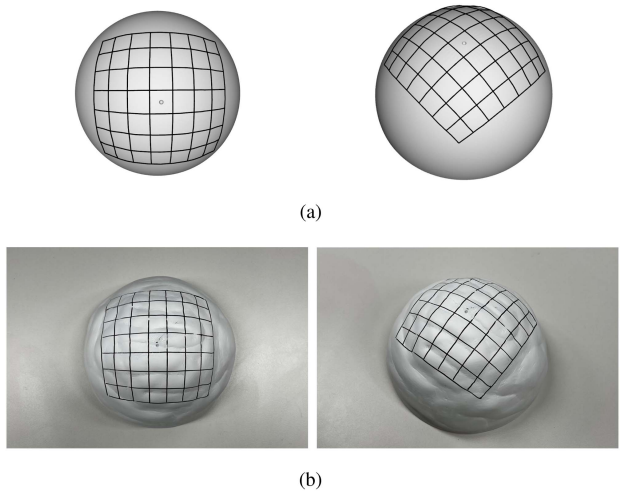


Fig. 13. Grid drawing experiment result on a half-sphere to show the effectiveness of the impedance control. Whereas the target drawing path is mapped on a clean half-sphere, the physical drawing is performed on a distorted half-sphere. (a) Target input drawing on a half-sphere. (b) Real robot drawing result on a distorted half-sphere.

2) Coverage Approximation: We made a few assumptions to approximate the set-cover problem in Section VI-B. First, we reduce the problem dimension from 3-D to 2-D by assuming that the canvas-surface geometry is a surface of extrusion. Based on this assumption, we find a set of coverage circles, which yields a suboptimal solution for the coverage problem. Our greedy set-cover algorithm adds to the suboptimality. However, the result is still conservative, meaning that we can draw all the strokes with a slightly higher number of robot-base placements. Second, the manipulator coverage map is discretized and precomputed. This only guarantees the reachability for a set of fixed drawing poses and not for the continuous drawing motion between poses. However, in practice, the discretization resolution of 0.05 m shows no problem in our drawing experiments. The redundancy of the manipulator also contributes to preventing failure. Finally, the coverage map can contain topological holes that may not be accurately approximated by a set of coverage circles. However, we can avoid reaching the holes by conservatively choosing the size of the coverage circles slightly larger.

3) Localization Error: Our drawing system uses IR tracking devices to localize the mobile base as well as to locate the canvas. As demonstrated in [58], the IR tracking device (the HTC Vive tracker) that we use for localization has a submillimeter precision. As a result of the robotic drawing experiment, a drawing error due to the localization was unnoticeable even though it may differ by the contents of the drawing.

IX. Conclusion

We have presented an artistic robotic pen-drawing system, SSK, that generates artistic pen art on large, nonplanar surfaces. We provide a pen interface for the artist to generate drawing strokes on a 2-D virtual canvas. Our system uses conformal mapping to project the 2-D drawing onto the 3-D canvas surface. Our navigation algorithm plans the motion for the mobile base to cover the large surface using the coverage map. The robotic drawing is performed using impedance control. We show a variety of drawing results using our proposed system. The results demonstrate that our system successfully brings the digital image into the real world and is not limited by the canvas size.

A. Limitations and Future Work

There are some limitations that we would like to address in our future work. Extending our set-cover planning method to 3-D to generate a more optimal solution is our immediate interest. One may use a convex polytope instead of a sphere or a circle to tightly represent coverage geometries [59]. Instead of the greedy algorithm, reformulating the problem for a mixed-integer program [60] could also be an option for generating more optimal solutions. Our reachability-based set-cover algorithm deals with only a discrete set of reachable poses, which does not guarantee continuous, reachable motion for a series of poses. Although all the drawing motions in our experiments were successfully carried out, validating the feasibility of the motion could be done in future work. Another interesting future direction is to use robot vision to reduce the drawing error. Although our drawing result is visually pleasing, it is difficult to quantify the artistic beauty reproduced by the robot in an objective manner. As a future research direction, it would be very interesting to replace the manual input drawing with an autonomous drawing, possibly using machine learning, so that our system can be evolved into a creative machine that generates an art piece by itself [26] or that collaborates with artists [61]. Finally, our system has great potential for application with other robotic systems that perform drawing tasks that require constant contact with a target surface while following a trajectory, such as robotic milling, sanding, or mopping.

References

[1] J. Reichardt, "Machines and art," *Leonardo*, vol. 20, no. 4, pp. 367–372, 1987.

[2] S. Calinon, J. Epiney, and A. Billard, "A humanoid robot drawing human portraits," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2005, pp. 161–166.

[3] T. Lindemeier, S. Pirk, and O. Deussen, "Image stylization with a painting machine using semantic hints," *Comput. Graph.*, vol. 37, no. 5, pp. 293–301, 2013.

[4] P. Tresset and F. F. Leymarie, "Portrait drawing by paul the robot," *Comput. Graph.*, vol. 37, no. 5, pp. 348–363, 2013.

[5] S. D. Herath and C. Kroos, *Robots and Art*. Singapore: Springer, 2016.

[6] O. Deussen, T. Lindemeier, S. Pirk, and M. Tautzenberger, "Feedback-guided stroke placement for a painting machine," in *Proc. 8th Int. Symp. Comput. Aesthetics Graph., Visualization, Imag.*, Annecy, France, 2012, pp. 25–33.

[7] T. Lindemeier, J. Metzner, L. Pollak, and O. Deussen, "Hardware-based non-photorealistic rendering using a painting robot," in *Computer Graphics Forum*, vol. 34, no. 2. Hoboken, NJ, USA: Wiley, 2015, pp. 311–323.

[8] B. Dalstein, R. Ronfard, and M. Van De Panne, "Vector graphics animation with time-varying topology," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 1–12, 2015.

[9] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, "A survey of robot manipulation in contact," *Robot. Auton. Syst.*, vol. 156, 2022, Art. no. 104224.

[10] N. Hogan, "Stable execution of contact tasks using impedance control," in *Proc. IEEE Int. Conf. Robot. Automat. Proc.*, vol. 4, 1987, pp. 1047–1054.

[11] F. J. Abu-Dakka and M. Saveriano, "Variable impedance control and learning–A review," *Front. Robot. AI*, vol. 7, 2020, Art. no. 590681.

[12] P. S. Heckbert, "Survey of texture mapping," *IEEE Comput. Graph. Appl.*, vol. 6, no. 11, pp. 56–67, Nov. 1986.

[13] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, "Least squares conformal maps for automatic texture atlas generation," *ACM Trans. Graph*, vol. 21, no. 3, 2002, pp. 362–371.

[14] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, 1998.

[15] D. Song, T. Lee, and Y. J. Kim, "Artistic pen drawing on an arbitrary surface using an impedance-controlled robot," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 4085–4090.

[16] D. Song and Y. J. Kim, "Distortion-free robotic surface-drawing using conformal mapping," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 627–633.

[17] P. McCorduck, *AARON'S CODE: Meta-Art, Artificial Intelligence, and the Work of Harold Cohen*. San Francisco, CA: W. H. Freeman, 1991.

[18] M. Stein and C. Madden, "The pumapaint project: Long term usage trends and the move to three dimensions," in *Proc. IEEE Intl Conf. Robot. Automat.*, 2005, pp. 2779–2784.

[19] J. M. Gülzow and O. Deussen, "Region-based approaches in robotic painting," in *Arts*, vol. 11, no. 4. MDPI, 2022, p. 77.

[20] L. Scalera, S. Seriani, A. Gasparetto, and P. Gallina, "Busker robot: A robotic painting system for rendering images into watercolour artworks," in *Proc. IFToMM Symp. Mechanism Des. Robot.*, Springer, 2018, pp. 1–8.

[21] L. Scalera, S. Seriani, A. Gasparetto, and P. Gallina, "Watercolour robotic painting: A novel automatic system for artistic rendering," *J. Intell. Robot. Syst.*, vol. 95, no. 3, pp. 871–886, 2019.

[22] A. Beltramello, L. Scalera, S. Seriani, and P. Gallina, "Artistic robotic painting using the palette knife technique," *Robotics*, vol. 9, no. 1, p. 15, 2020. [Online]. Available: https://www.mdpi.com/about/announcements/784

[23] L. Scalera, S. Seriani, P. Gallina, M. Lentini, and A. Gasparetto, "Human–robot interaction through eye tracking for artistic drawing," *Robotics*, vol. 10, no. 2, p. 54, 2021.

[24] T. Löw, J. Maceiras, and S. Calinon, "Drozbot: Using ergodic control to draw portraits," *IEEE Robot. Automat. Lett.*, vol. 7, no. 4, pp. 11728–11734, Oct. 2022.

[25] P. Schaldenbrand and J. Oh, "Content masked loss: Human-like brush stroke planning in a reinforcement learning painting agent," *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 1, pp. 505–512, 2021.

[26] P. V. Arman, "Cloudpainter: An artificially intelligent painting robot," 2018. [Online]. Available: http://www.cloudpainter.com/

[27] P.-L. Wu, Y.-C. Hung, and J.-S. Shaw, "Artistic robotic pencil sketching using closed-loop force control," *Proc. Inst. Mech. Eng., Part C: J. Mech. Eng. Sci.*, vol. 236, no. 17, pp. 9753–9762, 2022.

[28] R. Liu, W. Wan, K. Koyama, and K. Harada, "Robust robotic 3-D drawing using closed-loop planning and online picked pens," *IEEE Trans. Robot.*, vol. 38, no. 3, pp. 1773–1792, Jun. 2022.

[29] D. S. Arnon, "Topologically reliable display of algebraic curves," *SIGGRAPH Comput. Graph.*, vol. 17, no. 3, pp. 219–227, Jul. 1983. [Online]. Available: http://doi.acm.org/10.1145/964967.801152

[30] G. Taubin, "Distance approximations for rasterizing implicit curves," *ACM Trans. Graph.*, vol. 13, no. 1, pp. 3–42, Jan. 1994. [Online]. Available: http://doi.acm.org/10.1145/174462.174531

[31] A. Quint, "Scalable vector graphics," *IEEE Multimedia*, vol. 10, pp. 99–102, 2003.

[32] C. Loop and J. Blinn, "Resolution independent curve rendering using programmable graphics hardware," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1000–1009, 2005.

[33] M. J. Kilgard and J. Bolz, "Gpu-accelerated path rendering," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 172:1–172:10, Nov. 2012. [Online]. Available: http://doi.acm.org/10.1145/2366145.2366191

[34] A. Orzan, A. Bousseau, P. Barla, H. Winnemöller, J. Thollot, and D. Salesin, "Diffusion curves: A vector representation for smooth-shaded images," *Commun. ACM*, vol. 56, no. 7, pp. 101–108, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2483852.2483873

[35] D. Sỳkora, J. Buriánek, and J. Zára, "Sketching cartoons by example," in *Proc. SBM*, 2005, pp. 27–33.

[36] J.-D. Favreau, F. Lafarge, and A. Bousseau, "Fidelity vs. simplicity: A global approach to line drawing vectorization," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 120:1–120:10, Jul. 2016. [Online]. Available: http://doi.acm.org/10.1145/2897824.2925946
[37] K. Hormann, B. Lévy, and A. Sheffer, "Mesh parameterization: Theory and practice," in *Proc. ACM SIGGRAPH ASIA Courses*, pp. 1–115, 2007.
[38] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy, *Polygon Mesh Processing*. Natick, MA, USA: AK Peters/CRC, 2010.
[39] W. T. Tutte, "Convex representation of graphs," in *Proc. London Math. Soc.*, Oxford, U.K., 1960.
[40] A. Sheffer and E. D. Sturler, "Parameterization of faceted surfaces for meshing using angle-based flattening," *Eng. With Comput.*, vol. 17, no. 3, pp. 326–337, Oct. 2001. [Online]. Available: https://doi.org/10.1007/PL00013391
[41] S. Ren, Y. Xie, X. Yang, J. Xu, G. Wang, and K. Chen, "A method for optimizing the base position of mobile painting manipulators," *IEEE Trans. Automat. Sci. Eng.*, vol. 14, no. 1, pp. 370–375, Jan. 2016.
[42] S. Vafadar, A. Olabi, and M. S. Panahi, "Optimal motion planning of mobile manipulators with minimum number of platform movements," in *Proc. IEEE Int. Conf. Ind. Technol.*, 2018, pp. 262–267.
[43] S. Seriani, M. Seriani, and P. Gallina, "Workspace optimization for a planar cable-suspended direct-driven robot," *Robot. Comput.- Integr. Manuf.*, vol. 34, pp. 1–7, 2015.
[44] S. Seriani, P. Gallina, and A. Gasparetto, "A performance index for planar repetitive workspace robots," *J. Mechanisms Robot.*, vol. 6, no. 3, 2014, Art. no. 031005.
[45] N. Hogan, "Impedance control: An approach to manipulation," in *Proc. Amer. Control Conf.*, 1984, pp. 304–313.
[46] S. A. Schneider and R. H. Cannon, "Object impedance control for cooperative manipulation: Theory and experimental results," *IEEE Trans. Robot. Automat.*, vol. 8, no. 3, pp. 383–394, Jun. 1992.
[47] F. Caccavale, C. Natale, B. Siciliano, and L. Villani, "Six-DOF impedance control based on angle/axis representations," *IEEE Trans. Robot. Automat.*, vol. 15, no. 2, pp. 289–300, Apr. 1999.
[48] T. Wimbock, C. Ott, and G. Hirzinger, "Impedance behaviors for two-handed manipulation: Design and experiments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2007, pp. 4182–4189.
[49] J. Lee, P. H. Chang, and R. S. Jamisola, "Relative impedance control for dual-arm robots performing asymmetric bimanual tasks," *IEEE Trans. Ind. Electron.*, vol. 61, no. 7, pp. 3786–3796, Jul. 2014.
[50] J. Liang, R. Lai, T. W. Wong, and H. Zhao, "Geometric understanding of point clouds using laplace-beltrami operator," in *Proc. Comput. Vis. Pattern Recognit. IEEE Conf.*, 2012, pp. 214–221.
[51] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (PCL)," in *Proc. Robot. Automat., IEEE Int. Conf.*, 2011, pp. 1–4.
[52] O. Porges, R. Lampariello, J. Artigas, A. Wedler, C. Borst, and M. A. Roa, "Reachability and dexterity: Analysis and applications for space robotics," in *Proc. Workshop Adv. Space Technol. Robot. Automat.*, 2015.
[53] A. Makhal and A. K. Goins, "Reuleaux: Robot base placement by reachability analysis," in *Proc. 2nd IEEE Int. Conf. Robot. Comput.*, 2018, pp. 137–142.
[54] V. V. Vazirani, *Approximation Algorithms*. vol. 1, Berlin, Germany: Springer, 2001.
[55] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robot. Automat. Mag.*, vol. 19, no. 1, pp. 18–19, Mar. 2012.
[56] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *Proc. IEEE-RAS 15th Int. Conf. Humanoid Robots*, 2015, pp. 928–935.
[57] M. Tang, M. Lee, and Y. J. Kim, "Interactive hausdorff distance computation for general polygonal models," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 1–9, 2009.
[58] M. Borges, A. Symington, B. Coltin, T. Smith, and R. Ventura, "HTC vive: Analysis and accuracy improvement," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 2610–2615.
[59] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Trans. Robot.*, vol. 34, no. 3, pp. 586–601, Jun. 2018.
[60] D. Song et al., "Solving footstep planning as a feasibility problem using l1-norm minimization," *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 5961–5968, Jul. 2021.
[61] S. Chung, "Exquisite corpus," 2019. [Online]. Available: https://sougwen.com/project/exquisite-corpus

Daeun Song (Member, IEEE) received the B.S. degree in computer science and engineering in 2017 from Ewha Womans University, Seoul, South Korea, where she is currently working toward the Ph.D. degree in computer science and engineering.

Her research interests include robot path, motion planning, and computational geometry.

Jiyoon Park is currently working toward the B.S. degree in computer science and engineering with Ewha Womans University, Seoul, South Korea.

Her research interests include human–robot collaboration, robot path, and motion planning.

Young J. Kim (Senior Member, IEEE) received the Ph.D. degree in computer science from Purdue University, West Lafayette, IN, USA, in 2000.

He is an Ewha Fellow Professor of computer science and engineering with Ewha Womans University, Seoul, South Korea. Before joining Ewha, he was a Postdoctoral Research Fellow with the Department of Computer Science, the University of North Carolina at Chapel Hill, Chapel Hill, NC, USA. He has authored or coauthored more than 100 papers in leading conferences and journals in the areas of his interest. His research interests include interactive computer graphics, computer games, robotics, haptics, and geometric modeling.

Dr. Kim was the recipient of the Best Paper awards at the ACM Solid Modeling Conference in 2003, the International CAD Conference in 2008, and the HCI Korea Conference in 2016, and the Best Poster award at the Geometric Modeling and Processing Conference in 2006.