SurfMan: Generating Smooth End-effector Trajectories on 3D Object Surfaces for Human-Demonstrated Pattern Sequence

Radoslav Skoviera[1,*] Jan Kristof Behrens[1,*] Karla Stepanova[1]

***Abstract*— Specifying robot tasks for low-volume manufacturing scenarios is an open problem. The state-of-the-art robotic systems enable the application of smooth 2*D* paths to a 3*D* surface but assume that these paths are given by the product engineer. We extend this approach by a novel method for tool-path specification which produces smooth paths from noisy demonstrations. The user demonstrates only short patterns and selects a base path relative to an object in front of the robot, along which these patterns should be applied. The representation based on polynomials allows controlling the grade of the smoothness of the resulting tool-path. We generate robot trajectories that are parametrized to meet the use-case specific constraints and adhere to the robot's kinodynamic limits. We propose a set of measures to evaluate the quality of the generated curves and corresponding trajectories with respect to executability by a robot. The evaluation in simulation and real-robot experiments showed that the robot is able to reach up to 15.9% higher constant speed on tool paths generated by our system compared to unprocessed paths.**

***Index Terms*— Learning from demonstration, Surface manipulation, Curve synthesis, Path following, Contour detection**

I. Introduction

An important use-case for industrial robots is the surface manipulation (i.e., tasks where robots modifies the properties or shape of surfaces) of workpieces such as engraving, spray painting, milling, grinding, application of sealant or glue, etc. In many of these tasks, the end-effector is guided along a 3D trajectory that consists of regular patterns, while keeping constant distance from the manipulated 3D surface, and following constraints on the speed and orientation of the end-effector. In mass production, the automation of these use-cases is standard. For small batch sizes, however, the effort required using conventional robot programming tools makes the automation often prohibitively expensive.

There are several key conceptual and technical challenges which have to be solved in order to enable the creation of surface manipulation tool-paths. These are to: a) capture and process motion patterns demonstrated by the user; b) obtain a baseline path (e.g., from a CAD model, object outline detection, or interactively defined by a user) and align it to the workpiece in front of the robot c) synthesize a curve based on the baseline and a demonstrated pattern that is suitable for the execution by a robot; and d) generate motion plans to follow the synthesized curves and parametrize the trajectories to meet end-effector speed requirements.

[1]Czech Technical University in Prague, Czech Institute of Informatics, Robotics, and Cybernetics, radoslav.skoviera@cvut.cz, jan.kristof.behrens@cvut.cz, karla.stepanova@cvut.cz

∗ both authors contributed equally

Our approach to the curve synthesis and path generation, is inspired by computer graphics, where the ability to apply a given pattern along the selected contour is a standard problem called non-photorealistic rendering ([1]). These methods, albeit very inspiring, are not directly applicable to generate 6*D* tool paths for robots. They typically produce (*i*) raster graphics that are not easily transformed into smooth 3*D* paths, and (*ii*) do not consider tool orientations. There are very few works which deal with the question how to generate executable 6*D* paths on the surface of the real objects from 2*D* curves. Typically, these consider only planar tasks [2] or fully defined smooth 2*D* curves [3] as well as require knowledge of the object position and 3*D* model. To achieve high-quality robot trajectories exploiting task-specific tolerances on the end-effector pose together with constraining the Cartesian end-effector velocity we combine Descartes planner's global optimization approach [4] with custom constraints in time-parametrization tool TOPPRA [5].

The main contributions of this paper are:

1) A specification method for complex curves, their application to the 3*D* surface and execution using real robots while satisfying use-case dependent constraints on end-effector pose and velocity.
2) A set of measures to evaluate the quality of the generated curves and corresponding trajectories with respect to its executablity by a robot.
3) Evaluation of the introduced methods and measures within an integrated system (see Fig. 1) that includes user specification of the patterns via custom tool, semi-automatic definition of the baseline from RGB-D camera images and adjustment of the resulting curve via GUI.

Additional materials including code, videos and trial data are available on the project webpage http://imitrob.ciirc.cvut.cz/surftask.html.

II. Related Work

Our work is closely related to vision based generation of robotic trajectories, where vision methods are used to detect the surface of the object and the corresponding path for the robot and to robot motion planning and control, as special robotic planners and controllers have to be utilized to enable execution of smooth trajectories with custom constraints.

Vision based generation of robotic trajectories. In the real world, the 3*D* model of the object and its surface has to be taken into account. The 3*D* model can be either modelled based on the visual detection or loaded directly from CAD or other mesh library.

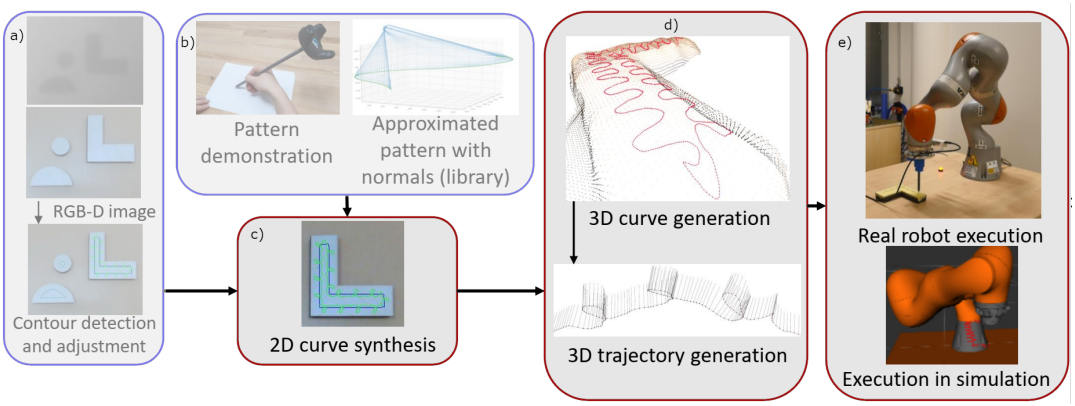


Fig. 1: Overview of the proposed methods as utilized in the application pipeline. The setup-dependent parts, a) and b), have blue border, the parts connected to the methods proposed in this paper have red border. The key parts are: a) Obtaining and aligning a baseline path (contour acquisition in our case), b) Processing of demonstrated patterns (here via demonstration tool utilizing an HTC Vive tracker), c) An algorithm to apply user-defined motion patterns along the baseline, d) A method for applying 2D paths on object surface to generate tool-paths, and e) A robot motion planner that takes a tool path and process-specific constraints.

In [6], the system for drawing pictures using a differential drive robot is described. They approximate edges detected by Canny edge detection with cubic B-spline. Their system has smaller state-space and is confined to the ground surface, whereas our 7-axis manipulator works on objects outside the table plane. In [7], a method for generating spray trajectory for automatic shoe sole spraying is proposed. B-spline interpolation and curve fitting are performed on the discrete contour points (acquired by edge detection from $2D$ image) to implement automatic generation of $3D$ spray trajectory. The curve generation is related to our method of detecting base contours. However, in none of these works, no synthesis of the final curve as a combination with the predefined patterns is allowed and no user interaction is considered. There are also commercial baking robots [2] which enable user specification of the cake decoration. Compared to our approach, only flat $2D$ patterns are allowed and the whole specification of the to be executed curve is necessary. Closely related work to our approach is [8], where they use a pen with a camera and a dotted paper to define paths for a welding robot. Compared to our system, fixed position of the object is expected, and the paths are fixed to one data source, which is 2D only and proprietary. We offer an algorithm that can work with any inputs (not only paths fixed to the CAD model), as long as they contain a series of poses. In our work, we take an inspiration from non-photorealistic rendering methods for curve synthesis. In order to apply the patterns along a curve we use a similar approach to [1]. The curves are treated as de-composable into multiple levels of details and feature extraction and reconstruction is performed using local transformations. We avoid the need for much data (that is needed e.g. in statistical approach to curve synthesis in the works of [9] and [10]) as our system works with a single demonstration. Furthermore, our approach only requires the user to demonstrate part of the desired trajectory.

Robot motion planning and control. Our sample surface manipulation task requires following toleranced $6D$ paths, i.e., where a tolerance for the end-effector position or orientation might be given. We need to calculate a joint-space motion that meets these requirements and also avoids self-collisions, joint-space discontinuities, and singularities. Nominally, the MoveIt Cartesian planner [11] could produce such trajectories, but due to its greedy algorithm it fails or returns plans of very low quality. In [12] a collision-free configuration-space path that closely follows a desired path in task space is produced using tools from computational geometry. [13] demonstrates spline path following for redundant robots. They have a 4-DoF robot and follow a $3D$ spline path. It is, however, not trivial to adapt these approaches to our 7-DoF robot and the tolerances could not be facilitated. In contrast, the Descartes planner [14] allows to incorporate also tolerances on the nominal path, which enables to optimize the robot motion. For example in [4], the Descartes planner was used for motion planning in welding use-cases. Descartes employs a brute-force approach and its computation and memory requirements make it scale badly. In this letter, we utilize the Descartes planner to derive distance optimized joint-space paths. Compared to the other approaches, we post-process these paths to satisfy also robot motion constraints and Cartesian velocity limits using custom constraints using TOPPRA [5], which is an algoritm based on reachability analysis enabling to create time-optimal plans.

III. Materials and Methods

In this section we describe the proposed methods to record and process custom patterns as well as to apply them along the given baseline curve.

A. Custom patterns specification and processing

We consider two types of patterns–(i) pure mathematical parametric curves, or (ii) patterns from a real demonstration consisting of set of poses along a trajectory. In the second

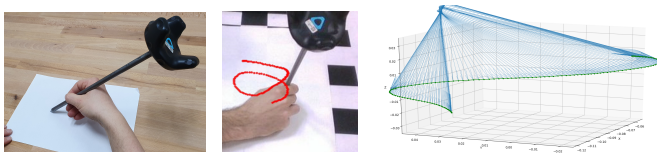


Fig. 2: Pattern definition using an HTC Vive motion tracker mounted on a pen-like device (left). The reprojection of recorded data to the camera image (middle). Trajectory with tool orientation (right).

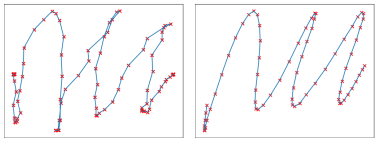


Fig. 3: Demonstrated pattern example. Left: unprocessed pattern; right: pattern after filtering and smoothing.

case, the demonstrated poses can be recorded in 2*D* (fixed z-coordinate and orientation), 3*D* (fixed orientation), or full 6*D* information (see Fig. 2). In this way, information about lifting or tool orientation can be utilized. In the following, we discuss the data formats, and the post-processing steps.

Post-processing. In case of demonstration, the recorded data is expected to be noisy with varying sampling rate, which can lead to irregularities and artifacts (see Fig. 3 left). Thus, the demonstrated patterns are not suited for further processing (e.g., B-spline construction fails). Therefore, the pattern is filtered from repetitive points, smoothed with a 1*D* Gaussian kernel, and regularly resampled (see Fig. 3 right). The filtering uses a simple approach. The distance between each two consecutive points is computed and if it is below a certain threshold, the second point is removed. This is done iteratively, until no close points exist.

Most robot controllers require that the jerk is limited and thus the acceleration is continuous. A curve with C^3 continuity can be traced with the end-effector moving at a constant speed (given that no joint limits are violated and the robot avoids singularities). There are many ways how to represent smooth curves [15]. In our case, approximation of the demonstrated, filtered, and smoothed patterns was done using polynomial curves, specifically *B*-splines. The amount of smoothening and number of points to resample are selected based on the preset threshold on the distance between the original and the smoothened curve. The user can adjust these values in GUI.

B. Pattern application along a baseline in 2D

The pattern application consists of merging a pattern with the baseline and generating a smooth curve. In the following we describe our approach that avoids a need for interpolation and enables smooth connection of individual patterns. Given a baseline *B*-spline B_c and a pattern *B*-spline B_p, the number of repetitions of the pattern I, the number of points which will be used for connecting consecutive patterns n_{gap}, the rotation of the pattern α, and its trimming from start (end) tr_s (tr_e) (see Fig. 7), we can generate the result curve Fig. 5a as described in the following.

To be able to shift and rotate the pattern points according to the baseline without the need for interpolation, we sample both baseline and the pattern in a way that there is a one-to-one mapping between baseline points $B_{c,s}$ and the repeated pattern points $B_{p,s}$, i.e.: $\text{length}\,(B_{c,s}) = I * [\text{length}\,(B_{p,s}) + 2n_{gap}]$. Rotation and trimming is applied to the pattern *B*-spline. The trimming removes the first tr_s and the last tr_e points from the sampled pattern *B*-spline $B_{p,s}$ (with length P): $B^t_{p,s} = \{B^i_{p,s} | tr_s \leq i \leq P - tr_e\}$. To enable smooth transition between patterns and scaling, we introduce a connection parameter n_{gap} that specifies the number of points which will be used to connect adjacent pattern repetitions (truncating n_{gap} points in the beginning and in the end of each pattern repetition). The connection between i-th and $(i+1)$-th pattern is the *B*-spline B^i_{gap}. Each pattern has $L = \frac{N}{I} - 2n_{gap}$ points, where N is the number of points on the baseline. The rotation and translation $R_i T_i$ to shift each pattern repetition $i \in \{1, \ldots, I\}$ around the contour is found as follows:

1. Using first two and last two points of the trimmed and rotated pattern, we create an approximate *B*-spline: $B_a = \text{bspline}(\{B^r_{p,s}(i) | i \in [0, 1, P-1, P]\})$.

2. For each pattern repetition $i \in \{1, \ldots, I\}$ we find the corresponding points on the contour $B_{c,s}$ and estimate a rigid transformation of the sampled approximate *B*-spline $B_{a,s}$ to these points (see Fig. 4b): $R_i T_i = \text{rigid\_trans}(B^i_{a,s}, B_{c,s}[iL, \ldots, (i+1)L - 1])$.

3. We apply the found transforms $R_i T_i$ to sampled points of the trimmed and rotated pattern $B^r_{p,s}$ for each $i \in \{1, ..., I\}$, j is indexing individual points of the pattern. The transformed patterns are superimposed on the underlying contour B_c as follows ($B_{c,r}$ being the resulting curve) (see Fig. 4c).:

$$B_{c,r}\left[i \cdot \frac{N}{I} + j\right] = R_i \cdot B^r_{p,s}[j] + T_i, \qquad (1)$$

$$\text{for } i \in [0, .., I-1], j \in 1, ..., \frac{N}{I}.$$

The tool orientation at each datapoint of the sampled pattern $B^r_{p,s}$ is rotated along the z axis by the rotation angle corresponding to the rotation matrix R_i. This is possible thanks to the assumption that the patterns are demonstrated on a flat surface.

4. *B*-spline B^{gap}_i for each connection between i-th and $(i+1)$-th pattern is estimated by exchanging last n_{gap} points of the i-th and first n_{gap} points from the $(i+1)$-th pattern by the sampled 7*D* bspline which was computed using 2 points before and after the gap (see Fig. 4e). The spline interpolation of positions and orientations enables smooth transition between end points of individual patterns. An example of the final curve is shown in Fig. 4f.

*C. Projection of the curve to 3*D *space*

Transferring the 2*D* curve into 3*D* space (see Fig. 5) is not a straightforward process. The solution has to also align the curve with a 3*D* surface, i.e., the workpiece. Our approach allows for two formats of inputs. Either a 3*D* model (e.g., a CAD model) of the object is provided in the form of a triangle mesh or a 3*D* point cloud (PCL), e.g. from a

(a) (b) (c) (d) (e) (f)

Fig. 4: Application of the pattern to the baseline including a gap. (a) baseline B_c. (b) Approximate *B*-spline B_a^i transformed around the baseline for each repetition i. (c) Pattern $B_{p,s}^r$ superimposed on the underlying baseline B_c via transform $R_i T_i$. (d) Final curve when no gap is used ($n_{gap} = 0$). (e) Applying *B*-spline B_i^{gap} to connect individual patterns. (f) Final curve when gap ($n_{gap} = 4$) is used.

(a) 2*D* curve (b) raw 3*D* (c) smooth 3*D* (d) up-sampled

Fig. 5: Transferring the curve from 2*D* to 3*D* and smoothing: (a) 2*D* curve resulting from application of a pattern to a baseline, (b) 3*D* curve fitted to the 3*D* *s*PCL. (c) smoothed 3*D* curve, (d) 3*D* curve computed from an up-sampled *s*PCL.

depth camera can be used, in case of a model-less workpiece alignment (as was in our test case, see Sec. IV). Since PCLs from depth cameras can be irregular and have holes (due to surface material properties and the sensing technique), as a first step, we use Poisson surface reconstruction [16] to obtain triangle mesh from the PCL (as shown in our application pipeline in the Section IV). In case of both data formats, the triangle mesh surfaces is then regularly sampled. We call this sampled surface *sampled point cloud* (*s*PCL).

The resulting *s*PCL is then masked to mark points on the surface of the workpiece that will be a part of the curve. To do this, a binary pixel mask is created from the 2*D* curve. The *s*PCL is then projected to the mask image space, resulting in a 2*D* *s*PCL, with each point having a correspondence in the 3*D* *s*PCL. For this, the mask and the *s*PCL has to be aligned. In case of *s*PCL computed from a sensed PCL from an RGB-D camera, the alignment is simple, since the camera image coordinate system coincides with the PCL, and thus the *s*PCL coordinates system. The masking of the *s*PCL then works as follows. For each 2*D* curve pixel ρ, all points ψ_i from the 2*D* *s*PCL (and thus also the 3*D* *s*PCL) are selected, for which the following holds:

$$||\rho - \psi_i||_2 \leq \tau,$$

where τ is a distance threshold. Each selected point ψ_i is assigned a weight based on the inverse of its distance from ρ. The final 3*D* curve point P, corresponding to ρ is computed as weighted average of the selected points ψ_i. The resulting 3*D* curve is smoothed by convolution with a Gaussian kernel.

Optionally, the user can set the z-axis (orthogonal to the table surface) position or offset for the 3*D* curve points. Finally, the surface normal for each 3*D* curve point is estimated from the surrounding *s*PCL points.

If custom tool orientation (from demonstration) should be applied, the normals are used to align it to the surface (Fig. 6d). The assumption is that the demonstration is done on a flat surface (e.g., a table), with the z-axis orthogonal to that surface. Thus, the surface normals for all points are aligned with the z-axis. During execution on a non-flat surface, the deviation of the normals from the world z-axis (orthogonal to the "main" plane, e.g., a worktable – typically aligned with the xy plane of the robot base coordinate system) is used to compute the adjustment for the tool orientation required to align it with the surface. Specifically, the following equation is used to compute a rotation vector from the world z-axis (z) and each normal n:

$$r_{vec} = \frac{z \times n}{\|z \times n\|} \arccos\left(\frac{z \cdot n}{\|z\| \, \|n\|}\right) \tag{2}$$

A rotation defined by the rotation vector r_{vec} is applied to the tool orientation. This is done for every point of the curve.

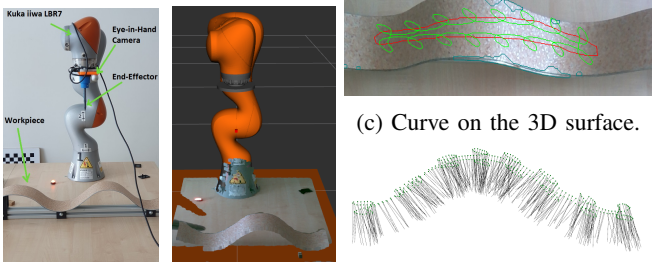
D. Robot Control

Executing a weakly constrained 6*D* path with a redundant manipulator efficiently is not a standard motion planning task. Furthermore, adhering to constraints on speed and orientation of the end-effector is making this optimization problem more tight. Here we describe our approach that introduces novel custom constraints used to time-parametrize joint-space paths produced by the Descartes planner.

The end-effector paths are specified by a sequence of points and tool orientations (see Fig.6d). The tool orientations can be surface normals of the target surface or custom tool orientation specified during the demonstration. We find a joint-space path by solving the path-wise inverse kinematics problem offline using the Descartes toleranced planner ([14], [4]) that showed in our initial evaluation better performance and reliability compared to Cartesian planner of the MoveIt motion planning framework.

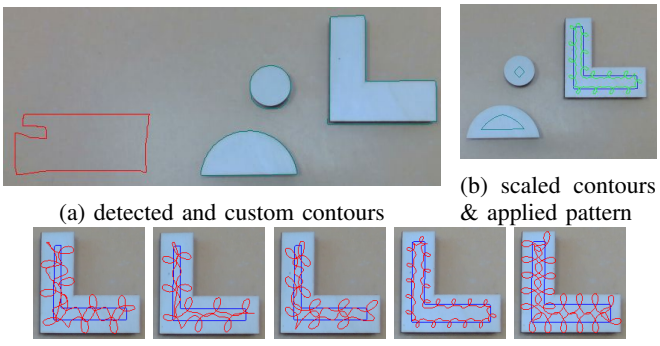
We resample the path to 250 points per meter in order to reduce the computational load and avoid numerical instabilities using *B*-spline interpolation. A toleranced trajectory point is created for every waypoint in the path (zero tolerance for the position and a 36° tolerance with 0.1° resolution around the z-axis of the end-effector). This makes the planner more robust against noisy input data (in the normals) and also allows optimizing the trajectory. It is important to select the path discretization in accordance with the tolerance resolution. Otherwise, the robot's dynamic limits might prevent to reach even the neighboring states or requires high accelerations to do so. Note that the tolerances are activity specific. For example, gluing, drawing, and milling have different requirements on the end-effector orientation.

We introduce custom constraints and time-parametrize the path that was returned by the planner using the TOPPRA algorithm [5]. In this way we create a trajectory that obeys the robot's joint velocity and acceleration limits and in parallel enforces the limit on the speed of the end-effector and minimizes the time for completing the motion. TOPPRA



(a) Real setup with 3D surface. (b) Detected pointcloud. (c) Curve on the 3D surface. (d) Input data for the planner (waypoints with normals).

Fig. 6: The object surface detected by Intel Realsense D435 camera (eye-in-hand) (a) is reconstructed (b), selected curve is applied to it (c) and the waypoints with adjusted tool orientations are sent to the robot (d).



(a) detected and custom contours (b) scaled contours & applied pattern (c) rotated pattern (d) rotated pattern (e) rotated pattern (f) no trimming (g) 20% trim

Fig. 7: (a) detected contours for all available objects visualised via GUI, a custom drawn contour (red). (b) A scaled contour with a pattern applied to it. Patterns can also be shifted, rotated ((c), (d), (e)), or trimmed (see (f) vs. (g)).

allows to specify constraints in the form of Eq. 3

$$\mathrm{F}(\mathbf{q})\mathrm{Dyn}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}) \leq \mathrm{g}(\mathbf{q}), \tag{3}$$

where Dyn is a function of the form given in Eq. 4:

$$\mathrm{Dyn}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}) = \mathrm{A}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^T\mathrm{B}(\mathbf{q})\dot{\mathbf{q}} + \mathrm{f}(\mathbf{q}). \tag{4}$$

By substituting $\mathrm{B}(q)$ with the product $J^T(q)J(q)$ ($J(q)$ being the end-effector Jacobian at $\mathbf{q}$), $\dot{\mathbf{q}}^T\mathrm{B}(\mathbf{q})\mathbf{q}$ equals the squared velocity of the end-effector. Setting $F(q) = I$, where I is the identity matrix, and $g(q) = \bar{v}^2$, we can formulate a general speed limit for the end-effector. To ensure numerical stability, we scaled the $g(q)$ and $\mathrm{Dyn}(q,\dot{q},\ddot{q})$ with a factor of 10^5.

A path-dependent speed limit on the end-effector can be formulated using the path-indexed version of the joint velocity constraint implemented in TOPPRA. It requires to provide a path-indexed velocity limit function–see our implementation of the function *vlims_func* in the code linked on our webpage. For every trajectory point, the Jacobian and the direction of motion are used to calculate an individual joint velocity limit. Note that this option allows a more flexible selection of the end-effector speed and is thus preferred.

We assume that the objects are largely convex, such that the surface points can be reached without collision when approached from the outside. We ensure that every path point is reachable by the robot. The user is notified about non reachable points and can adjust the object's position.

IV. Robotic and Experimental Setup

To test the proposed methods we created two physical setups. First one for custom pattern demonstration and second one for execution of the generated path on a real robot.

Pattern demonstration setup The demonstration setup consists of an HTC Vive virtual reality tracking system. An HTC Vive tracker, providing 6*D* poses at 60Hz, was mounted on the end of a pen-like metal rod (see Fig. 2). The other end (i.e., the "tip") of the rod was used to draw a pattern. The position of the tip of the rod was calibrated by fixing the tip in one place and performing a spherical motion with the end where the HTC Vive tracker was mounted. This generated points on a sphere with the center in the tip. The center was then estimated from the points using ordinary least squares optimization. For visualization purposes, there was also a camera calibrated towards the HTC Vive coordinate system. We record position and orientation of the tool, allowing demonstration of basically arbitrary pattern on a flat surface.

Robotic setup The real robot setup is shown in Fig. 6a. It consists of a KUKA iiwa LBR 7 robot with a Realsense D435 camera attached to the last link and a pen as end-effector (passively compliant in tool axis). The robot is controlled via ROS. We use the Descartes [14] and the MoveIt! Framework [11] for motion planning.

Experimental application pipeline For user interaction with the testing system, we developed a graphical user interface (GUI) which supports the baseline definition and alignment as well as pattern application functionality. The user can select a detected object contour or draw a custom one (see Fig. 7a and Fig. 7b), select pattern to apply to this contour and adjust various curve and execution parameters (see Fig. 7c-7g) before starting the execution by the robot. The applied pattern can be inspected in 2*D* and 3*D*. The contour detection is done from a robot-mounted camera.

V. Evaluation Measures

In this section, we present measures to evaluate the quality of the generated curves and corresponding trajectories with respect to their executability by a robot as well as the quality of the executed path. First, we want to demonstrate the quality of the produced curves on a purely geometric level (M_s). Second, an end-effector centric set of measures (M_v and M_d) to show quality of the task execution is presented. The third group of measures(M_{vm}, M_a, and M_p) concentrates on the cost associated with the robotic motion.

Measures M_v and M_d are dependent not only on the generated curve, but on the whole motion planning and control pipeline of the used robot. For the measures M_v, M_{vm}, and M_a we exclude the start (first acceleration) and

end (last deceleration) of the trajectory execution from the computation to avoid systematic error.

Smoothness of the curve (M_s): We consider similar smoothness measure of the generated curve as was used in [17], [18], namely the integral over the square of arc-length derivative of curvature along the path. M_s is an equivalent measure for discretized curves (constant sampling density), which gives a more detailed account over the distribution of the curvature.

The angle φ_i between the vectors spanned by each three consecutive points (x_i, x_{i+1}, x_{i+2}) is a measure for the curvature of the path (see Fig. 8).

$$\varphi_i = \arccos\left(\frac{<\bar{u}, \bar{n}>}{|\bar{u}|.|\bar{n}|}\right) < T_i \tag{5}$$

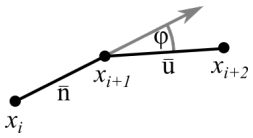


Fig. 8: Computation of the angle between three consecutive curve points.

We divide the range of $180°$ into five bins using the following threshold values: $T_0 = 0°$, $T_1 = 10°$, $T_2 = 30°$, $T_3 = 45°$, $T_4 = 90°$, and $T_5 = 180°$. We use the probability density function (PDF) approximated over these 5 bins is a measure for the smoothness:

$S_k = \frac{1}{n}\sum_{i=1}^{n}[|\varphi_i| \in [T_{k-1}, T_k]]$. The smoothness measure (M_s) is then characterized by the 5-tuple of these values:

$$M_s = \{S_1, S_2, S_3, S_4, S_5\}. \tag{6}$$

End-effector velocity deviation (M_v): Let v_i be the end-effector speed at time t_i. We approximate the integral over the squared deviation from the set speed v_0 by a finite sum (see Eq. 7), where $T = t_n - t_0$, $\Delta t_i = \frac{t_{i+1} - t_{i-1}}{2}$

$$M_v = \frac{1}{T}\int_o^T (v - v_0)^2 dt = \frac{1}{T}\sum_{i=1}^{n}(v_i - v_0)^2 \Delta t_i. \tag{7}$$

The maximum stable velocity (M_{vm}) is the maximum velocity v_0, for which the M_v (Eq. 7) is under a given threshold T:

$$M_{vm}^T = v_{max}^T = \underset{v_0}{\operatorname{argmax}}\, M_v \leq T. \tag{8}$$

Acceleration (M_a): As a measure for the wear and tear as well as the energy consumption, we report weighted mean of the absolute acceleration values for each joint.

$$M_a = \frac{1}{T}\sum_{i=1}^{n} \Delta t_i ||\mathbf{w}\mathbf{a}_i||_2, \tag{9}$$

where a_i^k is the acceleration of the *k-th* joint at the *i-th* point in the trajectory and w_k is a positive weight to scale the impact of the *k-th* joint. When choosing $w_k = (a_{\max}^k)^{-1}$, M_a becomes a measure for the saturation of the acceleration limits.

Length of a motion (M_p): We consider the (weighted) length of a motion in joint-space as a good measure to compare several motion plans for the same task (shorter is better). The used diagonally weighted norm for this measure allows to account for the different cost incurred by the motion of each joint (earlier joints in kinematic chains usually have to overcome significantly higher inertial moments). The measure is defined as:

$$M_p = \sum_{i=0}^{N-1} L_w(\mathbf{x_i}, \mathbf{x_{i+1}}, \mathbf{w}), \tag{10}$$

with L is defined for $a, b \in \mathbb{R}^n$ and unit vector $w \in \mathbb{R}_+^n$ as

$$L(\mathbf{a}, \mathbf{b}, \mathbf{w}) = \sqrt{\sum_{i=1}^{N} w_i (a_i - b_i)^2}. \tag{11}$$

Deviation from the reference curve (M_d): Dynamic Time Warping (DTW) [19] is used as a measure to evaluate the positioning precision of the end-effector during the motion relative to the reference curve. It is necessary to use DTW since no timing information is available for the reference curve. We use the *fastdtw* python implementation [20] with radius 30 and norm the result with the number of recorded joint states in the log $n = ||\log||$ in the execution:

$$M_d = \frac{1}{n}\mathrm{DTW}(\log, \mathrm{ref}) \tag{12}$$

VI. Experimental Results

In this section, we evaluate the quality of the generated trajectories as well as task execution quality, i.e., how well the robot (real or simulated) was able to perform a given task. The tasks require to move the end-effector with a constant (or given) speed along a path in space. We report the values for the proposed evaluation measures (as listed in Sec. V). We also show the effects of individual processing steps, different robot planners as well as real vs. simulated execution.

A. Pattern Processing

We evaluated the effect of pattern quality on the quality of the generated trajectory and its execution. We compared hand demonstrated pattern with a basic processing (only filtering – without it, the pattern was too noisy to convert to a B-spline), hand demonstrated pattern with full processing (filtering, smoothing and resampling), artificial pattern generated directly by sampling from a smooth *B*-spline, and no pattern. The experiments were conducted with a curve resulting from 30 pattern repetitions applied along an artificial circle baseline. The velocity deviation error (M_v) for different given speeds is shown in Fig. 9.

As expected, the least processed pattern, have worse deviations from the desired speed over the whole range of tested speeds. For the baseline experiment, i.e., when no pattern is applied, M_v is significantly lower than for all the other cases and thus the maximum stable velocity achievable

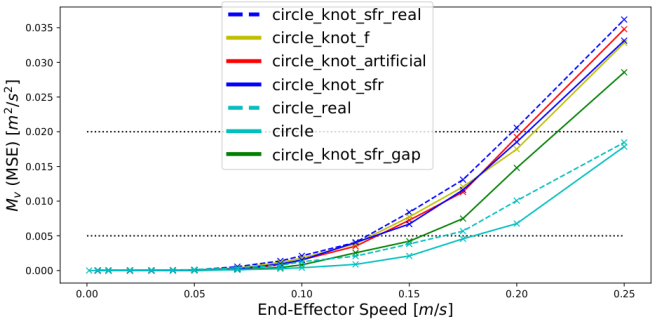


Fig. 9: End-effector velocity deviation M_v for varied v_0 for real and simulated executions of 10 pattern repetitions in different processing stages. The baseline is an artificially generated circle with diameter of 10 cm. The patterns are increasingly processed knots (only filtered - f, fully processed - sfr, fully processed with gap - sfr-gap, and artificial). Additionally, executions of a circle without pattern are shown.

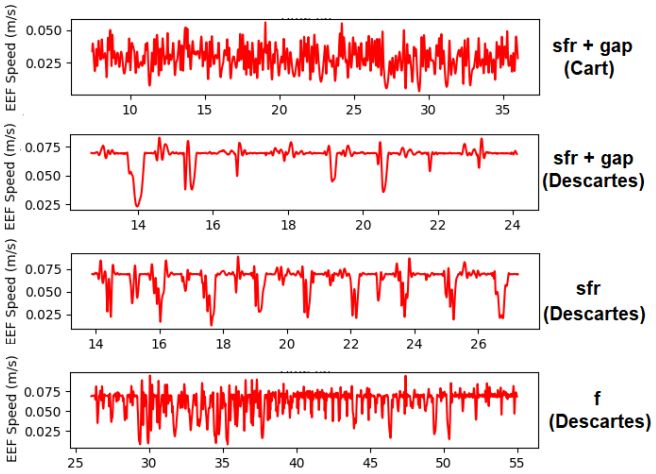


Fig. 10: Comparison of the end-effector speed over time for 10 knot repetitions on a circle contour with 5cm radius and end-effector set speed $v_0 = 0.07\frac{\text{m}}{\text{s}}$. Compared are: MoveIt! Cartesian planner with fully processed pattern with gap (*sfr +gap, cart*) compared to Descartes planner with only filtered pattern (f), fully processed pattern without gap (*sfr*), and fully processed pattern with gap (*sfr + gap*).

is significantly higher. Artificial and processed patterns reach similar M_v for lower speeds, but for speeds above 0.1 m/s the fully processed hand demonstrated patterns achieve lower M_v, i.e., are easier to execute.

Non-smooth patterns lead to micro oscillations saturating the actuators. To follow the path the robot must slow down until the acceleration limits are not violated anymore. The comparison of end-effector speeds over time for $v_0 = 0.07\,\text{m/s}$ (10 repetitions of knot applied to a circle) are visualised in the Fig. 10. The differences in velocity stability for various levels of processing are for this speed already clearly visible. The application of a gap connecting consecutive iterations of patterns via a fitted *B*-spline allows 14.1% faster executions at $M_v = 0.005\frac{\text{m}^2}{\text{s}^2}$ ($M_{vm}^{0.005}$ 15.3 vs. 13.4 $\frac{\text{cm}}{\text{s}}$) and 14.6% faster executions at $\frac{M_v=0.02\text{m}^2}{\text{s}^2}$ ($M_{vm}^{0.02}$ 23.5 vs. 20.3 $\frac{\text{cm}}{\text{s}}$) compared to fully processed path where no gap was applied. The executions on the real robot yielded to $M_{vm}^{0.005}$ 13.1 and 16.6 $\frac{\text{cm}}{\text{s}}$ for the fully processed and the circle without pattern, respectively.

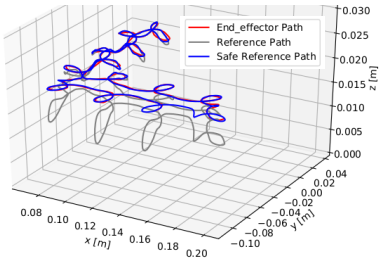
Application of the gap improves all of the measures (see Table I). The improvement of M_s^{S1+S2} measure indicates that the application of the gap avoids the typical sharp turns between the patterns (both $M_s^{S4} = 0$ and $M_s^{S5} = 0$), which affects the executability by the robot the most. The measure M_p can only be compared for the same pattern, i.e., the hand demonstrated patterns. It can be seen that application of gap significantly shortens the traveled distance as the connection between patterns is way smoother. Similar results were observed for other applied patterns. Fig. 12 gives visual examples of several hand drawn patterns applied to two different contours. See our webpage for more examples.

Pattern	M_s^{S1+S2}	$M_v[10^{-5}]$	M_a	M_p	$M_d[10^{-3}]$
none	1.00	40.0	0.427	2.14	N/A
knot A	0.97	153.6	0.795	4.26	2.91
knot (filt)	0.94	177.3	0.783	6.23	1.74
knot (fully)	0.95	150.1	0.793	6.38	1.91
knot(fully+gap)	**0.97**	**62.3**	**0.763**	**5.85**	**1.67**

TABLE I: Evaluation of measures for a circle contour with the following patterns (run on simulated robot): no pattern (none), with artificial knot (knot A), with fully processed demonstrated knot (knot fully), with only filtered knot (knot filt), and with fully processed knot with gap (fully+gap). The desired execution speed was $v_0 = 0.1$m/s.

B. Real vs. simulated experiments and comparison of robotic motion planners

To validate our approach including the simulation experiments, we implemented the system on a real robot shown in Fig. 6a. We conducted two types of experiments. The first experiment was conducted on the L-shaped workpiece where the knot pattern was applied with end-effector speed $v_0 = 0.05\ \frac{\text{m}}{\text{s}}$. Fig. 11a shows the path generated from the visual input (cf. also Fig. 5d) of the scene and the selection of a pattern in grey. For safety, we filter this path to avoid collisions with the table (i.e. enforcing a lower bound on the z-value). The resulting safe reference path is show in blue. The actual end-effector path is depicted in red. The precision of keeping the velocity constant was $M_v^{\text{real}} = 0.0012\frac{\text{m}^2}{\text{s}^2}$ and even slightly better than the simulated value of $M_v^{\text{sim}} = 0.0015\frac{\text{m}^2}{\text{s}^2}$ (see Fig. 9). The average acceleration of the joints was $M_a = 0.31\frac{\text{m}}{\text{s}^2}$, the travelled distance in joint space $M_p = 5.56$ rad, and values achieved by Cartesian planner were $M_v = 1.91 \cdot 10^{-5}\ \frac{\text{m}^2}{\text{s}^2}$, $M_d = 1.8 \cdot 10^{-3}$ m, $M_a = 0.34\ \frac{\text{m}}{\text{s}^2}$, and $M_p = 5.87$ rad. The velocity stability is with $M_v^{real} = 6.01197 \cdot 10^{-5}\frac{\text{m}^2}{\text{s}^2}$ better than the simulated $M_v^{sim} = 9.11 \cdot 10^{-4}\frac{\text{m}^2}{\text{s}^2}$. The inferior performance of the Cartesian planner compared to the Descartes planner w.r.t. the velocity stability for $v_0 = 0.07\frac{\text{m}}{\text{s}}$ is also nicely visible in the Fig. 10. Detailed comparison on various curves and pat-



(a) TCP curve request based on the camera images (grey), safety filtered (blue), and the actually measured TCP path (red).

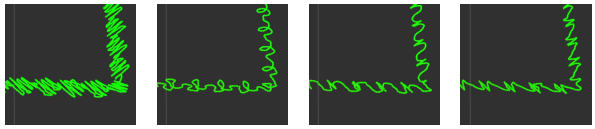


Fig. 12: Visual samples of the patterns *Zigzag*, *Handknot*, *S-wave*, and *Z-wave* on square (bottom) contours. Neither rotation nor trimming is applied.

terns of Cartesian and Descartes planner is on our webpage.

Additionally, we run the velocity stability experiment on a real robot (see Fig. 9). Same as in the simulation, the knot pattern was 10 times applied on the circle with a diameter of 10 cm. For velocities up to $0.05\frac{\mathrm{m}}{\mathrm{s}}$, the stability is very good (low error of $M_v \approx 0.001\frac{\mathrm{m}^2}{\mathrm{s}^2}$) for all experiments (see also Fig. 9). Above that, the real executions are slightly worse than the corresponding simulated ones.

VII. Conclusion and Discussion

In this paper we proposed methods for the specification and processing of surface manipulation tasks. The processing pipeline takes as an input user-defined patterns and baseline connected to the given surface and generates a tool-path by applying the pattern along the selected baseline. The stable end-effector speed required by many industrial tasks is enforced by custom constraint in the time parametrization process. We also present a set of measures which we used to evaluate the quality of the generated curves and corresponding tool-paths w.r.t. their executability by a robot.

The proposed methods were tested in several simulation experiments and also on a real robotic setup. The experiments have shown that our system is able to produce a smooth trajectory which is executable by a robot for a variety of patterns and baselines (see Fig. 12).The experiments also confirm the hypothesis that higher processing of the pattern results for this type of data in a smoother trajectory which allows execution of the path at a higher stable speed. Although for the speeds up to 0.05 cm/s the task can be executed with very stable velocity independently on the level of processing, for higher speeds we observed significant differences (see Fig. 9). M_v increases for higher speeds and less smooth tool paths as individual joint acceleration limits will become saturated. For example, the application of the gap to connect fully processed patterns resulted in approx. 15% faster executions at both $M_v = 0.005\frac{\mathrm{m}^2}{\mathrm{s}^2}$ and $M_v = 0.02\frac{\mathrm{m}^2}{\mathrm{s}^2}$ compared to the case when no gap is applied. The proposed methods can help in automation of tasks in small lot-size production scenarios. The pattern definition methods facilitate the task definition. The curve processing and trajectory parametrization makes the execution more efficient.

VIII. Acknowledgment

This work was supported by the European Regional Development Fund under project Robotics for Industry 4.0 (reg. no. $CZ.02.1.01/0.0/0.0/15\_003/0000470$), MPO TRIO project num. FV40319, and by the Czech Science Foundation (project no. GA21-31000S).

References

[1] A. Hertzmann, N. Oliver, B. Curless, and S. M. Seitz, "Curve analogies," *Eurographics Workshop on Rendering*, p. 14 pages, 2002.

[2] Unifiller, "Baker-bot robot cake cookie and pastry decorator — unifiller." [Online]. Available: https://www.unifiller.com/bakery-machine/baker-bot

[3] C. Hartl-Nesic, T. Glück, and A. Kugi, "Surface-based path following control: Application of curved tapes on 3-d objects," *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 615–626, 2021.

[4] J. D. Maeyer, B. Moyaers, and E. Demeester, "Cartesian path planning for arc welding robots: Evaluation of the descartes algorithm," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep 2017, p. 1–8.

[5] H. Pham and Q. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, p. 645–659, Jun 2018.

[6] C.-L. Shih and L.-C. Lin, "Trajectory planning and tracking control of a differential-drive mobile robot in a picture drawing application," *Robotics*, vol. 6, no. 3, p. 17, 2017.

[7] T. Wang, J. Zhang, J. Dong, S. Pan, and L. Zheng, "A method for generating spray trajectory of a shoe sole based on laser vision," in *IEEE ICIVIC*. IEEE, 2019, pp. 36–39.

[8] J. Pires, T. Godinho, and R. Araújo, "Using digital pens to program welding tasks," *Industrial Robot: An International Journal*, vol. 34, no. 6, p. 476–486, Oct 2007.

[9] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein, "WYSIWYG NPR: Drawing strokes directly on 3D models," *ACM Trans. on Graphics*, vol. 21, no. 3, pp. 755–762, July 2002.

[10] S. Simhon and G. Dudek, "Sketch interpretation and refinement using statistical models." in *Rendering Techniques*, 2004, pp. 23–32.

[11] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[12] R. Holladay, O. Salzman, and S. Srinivasa, "Minimizing task-space frechet error via efficient incremental graph search," *IEEE RAL*, vol. 4, no. 2, pp. 1999–2006, 2019.

[13] R. J. Gill, D. Kulić, and C. Nielsen, "Spline path following for redundant mechanical systems," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1378–1392, 2015.

[14] *Descartes Motion Planner*. ROS-Industrial Consortium. [Online]. Available: https://github.com/ros-industrial-consortium/descartes

[15] P. Gallina and A. Gasparetto, "A technique to analytically formulate and to solve the 2-dimensional constrained trajectory planning problem for a mobile robot," *Jour. of Intelligent and Robotic Sys.*, vol. 27, no. 3, pp. 237–262, 2000.

[16] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proc. of Eurographics Symp. on Geo. proc.*, vol. 7, 2006.

[17] Y. J. Kanayama and B. I. Hartman, "Smooth local-path planning for autonomous vehicles1," *IJRR*, vol. 16, no. 3, pp. 263–284, 1997.

[18] T. Berglund, A. Brodnik, H. Jonsson, K. Mrozek, M. Staffanson, and I. Söderkvist, "Planning smooth paths among obstacles using minimum curvature variation B-splines."

[19] M. Müller, *Information Retrieval for Music and Motion*. Springer Berlin Heidelberg, 2007. [Online]. Available: http://link.springer.com/10.1007/978-3-540-74048-3

[20] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, p. 561–580, Oct 2007.