Can Robots Mold Soft Plastic Materials by Shaping Depth Images?

Ege Gursoy, Sonny Tarbouriech, Andrea Cherubini

This article has been accepted for publication in IEEE Transactions on Robotics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TRO.2023.3288836

Can robots mold soft plastic materials by shaping depth images?

Ege Gursoy, Sonny Tarbouriech, Andrea Cherubini

***Abstract*—Can robots mold soft plastic materials by shaping depth images? The short answer is no: current day robots can't. In this article, we address the problem of shaping plastic material with an anthropomorphic arm/hand robot, which observes the material with a fixed depth camera. Robots capable of molding could assist humans in many tasks, such as cooking, scooping or gardening. Yet, the problem is complex, due to its high-dimensionality at both perception and control levels. To address it, we design three alternative data-based methods for predicting the effect of robot actions on the material. Then, the robot can plan the sequence of actions and their positions, to mold the material into a desired shape. To make the prediction problem tractable, we rely on two original ideas. First, we prove that under reasonable assumptions, the shaping problem can be mapped from point cloud to depth image space, with many benefits (simpler processing, no need for registration, lower computation time and memory requirements). Second, we design a novel, simple metric for quickly measuring the distance between two depth images. The metric is based on the inherent point cloud representation of depth images, which enables direct and consistent comparison of image pairs through a non-uniform scaling approach, and therefore opens promising perspectives for designing *depth image – based* robot controllers. We assess our approach in a series of unprecedented experiments, where a robotic arm/hand molds flour from initial to final shapes, either with its own dataset, or by transfer learning from a human dataset. We conclude the article by discussing the limitations of our framework and those of current day hardware, which make human-like robot molding a challenging open research problem.**

***Index Terms*—Robotic Manipulation, Vision-Based Control, RGB-D Image Processing, Machine Learning.**

I. INTRODUCTION

THROUGHOUT evolution, humans have assimilated amazing manipulation skills. An example is sculpting, which has progressed from prehistoric cave handprints to Renaissance masterpieces (see Fig. 1[1]). Achieving these results requires many capabilities. First, the sculptor must conceive a rough representation of the intermediate shapes, leading from initial to desired state of the molding material. Then, s/he must choose and use the most appropriate actions and tools for obtaining each of these intermediate shapes. While sculpting, s/he relies on perception (mostly visual and tactile) of the material properties and shape, to continuously update the process. Similar, albeit simpler, skills are needed for everyday tasks, such as gardening, cooking, massaging or manipulating cloak.

Recently, [1], [2] have presented impressive robotic sculpting, achieved with a dual arm manipulator cutting clay via a hot wire. Other pioneer works on robotic sculpture include [3], [4]. While their approaches are very original, the authors of these works focus on a specific task (sculpting), and rely on tools (e.g., hot wire) customized for their application and matter (e.g., clay). In the authors' view, molding in human-like manner soft materials would enable robots to perform a plethora of new operations and assist humans in many daily chores, well beyond robotic sculpture [1]-[4]. However, to date, roboticists have mainly focused on rigid object manipulation. The reason is that manipulating soft materials requires perceiving and modeling deformations at a high frame rate. While visual features can be consistently detected and tracked on rigid objects, they change over time on deformable materials, misleading both model-based and feature-based visual trackers. Tactile and force feedback could also be beneficial. These senses could complement the 3D geometry measured by vision, to infer the material's mechanical properties. Yet, complicated contact models are necessary to map force/tactile signals to the corresponding object surface displacements. Furthermore, the force/tactile measures should cover, with sufficiently high resolution, the whole robot skin. To the best of our knowledge, to date there exists no machine with such actuation and sensing capabilities.

Fig. 1: It took over 20'000 years for human sculpture to evolve from the prehistoric cave finger flutings of Gargas in France (left) to Michelangelo's Moses (right).

In our previous work [5], we have addressed kinetic sand molding with 2D vision. Yet, without depth information, the set of controllable actions/effects is reduced. Typically, in [5], we could only modify the material's visible contours. That experience convinced us to opt, here, for depth vision, which – compared to 2D vision – enables shaping along all three directions. Besides, depth cameras, such as the Microsoft

All authors are with LIRMM, Univ. Montpellier, CNRS, Montpellier, France. firstname.lastname@lirmm.fr

[1]Figure sources: https://en.wikipedia.org/wiki/File:GargasFlutings.jpg and https://www.michelangelo.org/images/artworks/moses.jpg

Kinect and the Intel RealSense, have recently gained a huge success in robotics.

Therefore, here, we explore the possibility of deforming soft plastic materials[2] by relying solely on depth images (i.e., with neither a model of the material, nor tactile feedback, nor registered point clouds). We propose the direct use of raw depth images instead of point clouds, to avoid cumbersome processing involving registration, and requiring high computational times and memory usage. Our rationale is similar to the one which motivated visual servoing directly in the image, rather than in the 3D space [6].

A. Related Work

Researchers have used point clouds to detect objects in indoor scenes [7], and to estimate their 6-DOF pose using 3D Geometric and Photometric feature descriptors [8]. A related application is semantic object labelling, realized in [9] with deep learning. These techniques require rich datasets, such as the Yale-CMU-Berkeley real-life objects set [10], which is specifically designed for benchmarking manipulation research. Crowd-sourcing can be useful to build and label point cloud datasets, as shown in [11]. Another application of point clouds is robot grasping, both with hands and suction cups. Pas and Platt [12] propose to detect grasp points on novel objects presented in clutter, while accounting for the robot hand geometry. The authors of [13] have introduced Dex-Net 3.0, a dataset of 2.8 million point clouds, suction grasps, and grasp robustness labels, which computes the quality of the seal between suction cup and local object surface. Point clouds are studied well beyond robotics, with applications including plant phenotyping [14].

Despite this success, point cloud dimensions make processing cumbersome: even most current day implementations of the Iterative Closest Point (ICP) algorithm [15] for aligning two point clouds are too slow for robot feedback control. ICP has been originally designed and applied to rigid scenes [16], with improved versions, as in [17], where translation and rotation are decoupled, and a fast Bound and Branch algorithm globally optimizes the 3D translation parameter first. Recently, [18] has proposed a fast (milliseconds) and certifiable algorithm for the registration of two point clouds in the presence of numerous outlier correspondences. The authors of [19] have extended point registration to non-rigid clouds, via a robust transformation learning scheme. The principle is to iteratively establish point correspondences and learn the nonrigid transformation between two given sets of points. To this end, they cast the point set registration into a semi-supervised learning problem.

At a higher level, many efforts have been put into non-rigid object tracking. DynamicFusion [20] is a dense simultaneous localization and mapping framework capable of reconstructing non-rigidly deforming scenes in real-time, by fusing RGB-D images. Since it does not require a template or other prior scene model, the approach is applicable to a wide range of moving objects and scenes. Other model-free approaches are: [21], which tracks – using touch and vision – the surface of a deformable object to be grasped by a robot, and [22], where a neural network can visually segment a pizza dough. An elegant alternative [23] consists in including – if they are known – the physical properties of the object, to track it from a sequence of point clouds, with a probabilistic generative model.

When it comes down to shaping the object, the robot should be capable of predicting the next state given the current state and a proposed action. To this end, [24] use some defined primitive tool action for rearranging dirt and [25] presents a Convolutional Neural Network for scooping and dumping granular materials. The authors of [26] learn a particle-based simulator for complex control tasks; this simulator adapts to new environments or to unknown dynamics within few observations.

B. Contributions

In this paper, we consider a manipulator equipped with an anthropomorphic hand and capable of applying three different actions at various positions of some plastic material, while observing it through a fixed depth camera. The robot must mold the material into a desired shape, by finding and applying the correct sequence of actions at the right positions. To find the correct sequence, the robot should predict the effect of each action on the environment, which is represented by a depth image. We rely on some realistic hypotheses, to make the prediction problem tractable, so that it breaks down to replacing local rectangular patches within the depth image. We generate the patches consequent to action execution, with three alternative data-based methods, two with *encoder-decoder image generator network*, and a simpler one with *difference compensation*. We train all three prediction methods with a robot dataset and with a human dataset, and we design a novel scalar metric for assessing the methods' precision, in terms of distance between point clouds. We assess our method in a series of experiments, where a robot molds desired shapes, either with its own dataset, or by *transfer learning* from the human dataset. Finally, we show that our method for finding the correct sequence of actions could generalize across actions, materials, and geometry.

The contributions of our paper are the following:

1) We propose an *original framework for depth image – based robot shaping of un-modeled plastic material*. Given an initial and a desired point cloud of the material, the robot plans and executes a sequence of different actions at various positions to mold the material. Due to its high dimensionality, the problem of predicting posterior (consequent to robot actions) point clouds, without a model of the material, would be intractable without an original idea: we show that, under reasonable assumptions, the shaping problem can be mapped from point cloud to depth image space, with the following benefits.
   - Point cloud registration and matching are not required anymore,
   - the lower needs in terms of computation time and memory usage make the action effect prediction and action planning sub-problems tractable,

[2]Plastic materials remain deformed after the external force is removed.

- standard image processing techniques as well as deep learning can be applied directly to the depth images.

2) To attain depth image – based shaping, we define a scalar metric for measuring the *distance between two point clouds*, in depth image space. The proposed distance metric is based on the observation that depth images inherently encode a point cloud, allowing for the direct comparison of image pairs in a consistent manner. In our work, we use this metric: *(a)* as the loss function of the generator which models changes produced by a robot action, *(b)* as a measure for planning the sequence of actions leading from initial to final point clouds, and *(c)* to assess the success of a point cloud shaping task. Since it is the weighted mean value of the difference between two 2D images, this metric can be computed very quickly, and with low memory needs, opening promising perspectives for designing depth image – based robot controllers.
3) We assess the framework in a *series of unprecedented experiments* with a robotic arm + hand, molding flour from initial to final shapes, by relying on both a robot and a human dataset. By exploiting transfer learning from the human, we do not need numerous robot experiments as in [25]. Furthermore, in contrast with similar works [24], [25], [26], by using a anthropomorphic robot, we push robotics one step closer to human-like molding. Yet, robotic molding remains an open problem, due to the many limitations of current day robots, emphasized in our experiments.

II. Modeling Depth Images and Point Clouds

In this section, we recall the fundamental notions of depth image theory, required in our work.

The *luminance* of each pixel of a depth camera of resolution $w \times h$ and quantification b bits is:

$$L_{uv} \in \mathcal{L} = \left\{0, 1, \ldots, 2^b - 1\right\} \subset \mathcal{N} \tag{1}$$

with $(u, v) \in \mathcal{I} = \{1, \ldots, w\} \times \{1, \ldots, h\} \subset \mathcal{N}^2$ the pixel coordinates. The entire depth image is the matrix:

$$\mathbf{L} = \begin{pmatrix} L_{11} & \cdots & L_{1w} \\ \vdots & \ddots & \vdots \\ L_{h1} & \cdots & L_{hw} \end{pmatrix} \in \mathcal{L}^{w \times h}. \tag{2}$$

In the following, we refer to *depth images* simply as *images*.

We express point coordinates in meters in the *camera frame* (center at the camera optical center, X increasing with the pixel columns, Y increasing with the pixel rows, Z coincident with the optical axis, and positive for points in front of the camera). Assuming a linear depth model, for any point $\mathbf{p} = (XYZ)^\top$ seen in the depth image at (u, v), luminance L_{uv} can be converted to Z:

$$Z\left(L_{uv}\right) = \frac{Z_{min} - Z_{max}}{2^b - 1} L_{uv} + Z_{max} \in \left[Z_{min}, Z_{max}\right]. \tag{3}$$

This requires knowing the camera depth range $[Z_{min}, Z_{max}] \subset \mathcal{R}^+$. Note that this model clips at Z_{max} (respectively, Z_{min}) the depths of all points which have depth greater (smaller) than Z_{max} (Z_{min}); the corresponding pixels are black (white), with $L_{uv} = 0$ ($L_{uv} = 2^b - 1$). We consider un-distorted perspective projection (i.e., a pinhole camera model). Then, the two other camera frame coordinates of $\mathbf{p}$ can be derived from Z as:

$$X\left(u\right) = \frac{u - u_0}{f_x} Z \text{ and } Y\left(v\right) = \frac{v - v_0}{f_y} Z \tag{4}$$

with $(u_0, v_0) \in [1, w] \times [1, h] \subset \mathcal{R}^2$ the camera principal point coordinates in the image plane and $(f_x, f_y) \in \mathcal{R}^2$ the camera focal lengths expressed in pixels. All points visible in the depth image belong to the camera *viewing frustum* $\mathcal{F} \subset \mathcal{R}^3$, which is a truncated pyramid lying between Z_{min} and Z_{max}.

Let us define $\mathbf{m}$ the vector function which combines (3) and (4) to map a pixel and its luminance to the camera frame coordinates of the corresponding point:

$$\begin{array}{rccc} \mathbf{m}: & \mathcal{I} \times \mathcal{L} & \rightarrow & \mathcal{F} \\ & \begin{pmatrix} u \\ v \\ L_{uv} \end{pmatrix} & \mapsto & \begin{pmatrix} X\left(u\right) \\ Y\left(v\right) \\ Z\left(L_{uv}\right) \end{pmatrix} \end{array} \tag{5}$$

Note that this mapping is *bijective*:

$$\begin{array}{rccc} \mathbf{m}^{-1}: & \mathcal{F} & \rightarrow & \mathcal{I} \times \mathcal{L} \\ & \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} & \mapsto & \begin{pmatrix} f_x X/Z + u_0 \\ f_y Y/Z + v_0 \\ \left(2^b - 1\right) \frac{Z - Z_{max}}{Z_{min} - Z_{max}} \end{pmatrix}. \end{array} \tag{6}$$

This is not the case of standard cameras, which cannot measure the point depth.

Definition: visible point cloud. Let us now represent the depth image as a set $\mathcal{D}$ of $w \times h$ triplets, containing the coordinates of each pixel of $\mathbf{L}$ along with its luminance:

$$\mathcal{D} = \left\{\left(u_i, v_i, L_{u_i v_i}\right)^\top, i = 1, \ldots, w \times h\right\} \subset \mathcal{I} \times \mathcal{L}. \tag{7}$$

The visible point cloud $\mathcal{P} = \{\mathbf{p}_1, \ldots \mathbf{p}_{w \times h}\} \subset \mathcal{F}$ corresponding to $\mathbf{L}$ is the *image* (in mathematical terms) of $\mathcal{D}$ under $\mathbf{m}$:

$$\mathbf{m}: \quad \mathcal{D} \quad \rightarrow \quad \mathcal{P} \tag{8}$$

Note that $\dim \mathcal{P} = w \times h$. Note also that not all points of $\mathcal{P}$ exist physically. In particular, points corresponding to $L_{uv} = 0$ and to $L_{uv} = 2^b - 1$ are artefacts due to the limited camera range. Furthermore, two points in $\mathcal{P}$ cannot be on the same optical ray (or else, one will occlude the other):

$$\forall \left(\mathbf{p}_i, \mathbf{p}_j\right) \in \mathcal{P} : \nexists k \in \mathcal{R} : \mathbf{p}_i = k \mathbf{p}_j. \tag{9}$$

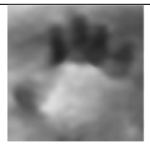
In this paper, we will also use normalized pixel coordinates:

$$x = \frac{u - u_0}{f_x} \quad \text{and} \quad y = \frac{v - v_0}{f_y}, \tag{10}$$

to rewrite equation (4) as:

$$X = xZ \quad \text{and} \quad Y = yZ. \tag{11}$$

TABLE I: Actions used in this work, with their characteristics.

Action $\mathbf{a}^i$	Description	Illustration	Effect on L	ΔX (mm)	ΔY (mm)	ΔZ (mm)
Grasp $\mathbf{a}^g$	The hand initially in claw shape is closed.			218	195	50
Knock $\mathbf{a}^k$	The closed fist pushes downward.			84	75	50
Press $\mathbf{a}^r$	The fist bottom pushes downward.			46	103	50
Pinch $\mathbf{a}^n$	The index and middle fingers close.			42	85	50
Poke $\mathbf{a}^p$	The index finger pushes downward.			22	18	50

III. Problem Statement

A. The point cloud shaping task

Consider a *robot*, observing via a fixed depth camera the *environment*. The robot can execute – on the point cloud $\mathcal{P}$ representing the environment – an action $\mathbf{a}$.

Definition: Robot action $\mathbf{a}$. An action is a set of joint space trajectories, which makes the robot modify $\mathcal{P}$. Each action $\mathbf{a}^i$ is applicable at a position $\mathbf{t} = (XYZ)^\top$ in the camera frame, within a finite set of possible (action-dependent) positions, $\mathbf{t}^j$, $j = 1, \ldots, N$. We name $\mathcal{A} = \left\{\mathbf{a}^{ij}\right\}$ the set of all actions over all positions.

In this work, we have selected 5 actions (grasp, knock, press, pinch and poke) after having inspected the database of human sculpting images presented in [5]. Table I shows the characteristics of each of these actions: name, description, illustration (when applied by humans), effect on a depth image $\mathbf{L}$, and "size" (parameters ΔX, ΔY, ΔZ, which we will detail hereby).

We model the robot and environment (i.e., the point cloud) as a discrete-time system. At each iteration $k \in [1, K] \subset \mathcal{N}$, the robot executes an action $\mathbf{a}_k \in \mathcal{A}$ on point cloud $\mathcal{P}_k$. The joint space trajectories of the action are completed within the iteration, and the point cloud is changed to a new point cloud

$$\mathcal{P}_{k+1} = \mathbf{f}\left(\mathcal{P}_k, \mathbf{a}_k\right), \tag{12}$$

with $\mathbf{f}\left(.\right)$ an appropriate mathematical function modeling the point cloud changes.

Hypotheses We rely on the following assumptions:

1) The process is Markov; therefore, we can remove the dependency from k and rewrite (12) as:

$$\mathcal{P}_{\mathbf{a}} = \mathbf{f}\left(\mathcal{P}, \mathbf{a}\right). \tag{13}$$

This assumption also implies that: 1/ no deforming phenomena, aside from those caused by $\mathbf{a}$, act on the point cloud, and 2/ the deforming phenomena caused by $\mathbf{a}$ are constant.

2) The effect of each action on the point cloud is bounded in the 3D space, and contained within a box, denoted $\mathcal{B}_{ij}$ for action $\mathbf{a}^{ij}$. We have designed the actions in $\mathcal{A}$, so that all boxes $\mathcal{B}_{ij}$ are entirely visible (i.e., $\mathcal{B}_{ij} \subset \mathcal{F}$). We formalize this hypothesis as follows: the points which differ between the prior and posterior point clouds are:

$$\begin{aligned} &\mathbf{p} \in \left(\mathcal{P} \cup \mathcal{P}_{\mathbf{a}_{ij}}\right) \setminus \left(\mathcal{P} \cap \mathcal{P}_{\mathbf{a}_{ij}}\right) \\ &\implies \mathbf{p} \in \mathcal{B}_{ij} = [X^-; X^+] \times [Y^-; Y^+] \times [Z^-; Z^+], \end{aligned} \tag{14}$$

with $\left(X^\pm Y^\pm Z^\pm\right)^\top = \mathbf{t} \pm \left(\Delta X \Delta Y \Delta Z\right)^\top$. The box is centered at the action's position $\mathbf{t}^j$, and its size depends on the action type (e.g., an action may affect a bigger volume than another, see Table I).

3) The positions of all actions in $\mathcal{A}$ have the same Z coordinate, and the depth of $\mathcal{B}_{ij}$ is negligible with respect to the action depth ($\Delta Z << Z$), so $Z^- \approx Z^+ \approx Z$.
4) The effects of the actions are position invariant: given i, the effect of $\mathbf{a}^i$ on the points contained in $\mathcal{B}_{ij}$ is the same for all $\mathbf{t}_j$, $j = 1, \ldots, N$.

Definition: point cloud shaping task Given a final desired point cloud $\mathcal{P}_*$, *the point cloud shaping task* consists in shaping point cloud $\mathcal{P}$ into $\mathcal{P}_*$. We define $d_k\left(\mathcal{P}_k, \mathcal{P}_*\right) \in \mathcal{R}^+$ as a scalar function, which measures the distance between $\mathcal{P}_k$ and $\mathcal{P}_*$, so that $d_k = 0 \iff \mathcal{P}_k = \mathcal{P}_*$. Then, the task of shaping a point cloud to $\mathcal{P}_*$ with an accuracy $\bar{d} \geq 0$, consists in applying a finite sequence of actions within $\mathcal{A}$: $\mathbf{s}_* = \{\mathbf{a}_1, \ldots, \mathbf{a}_K\}$ so that after K iterations: $d_K \leq \bar{d}$. We tolerate such an upper bound on the accuracy, since even a human would be incapable of perfectly reproducing ($d_K = 0$) a given point cloud. Yet, the sequence of actions should make d_K decrease. More formally, it must be possible to apply, at each iteration k, an action $\mathbf{a}_k$ to reduce d_k:

$$d_{k+1} < d_k. \tag{15}$$

The above equation assumes that the task can be solved using a greedy algorithm. Greedy heuristics are known to

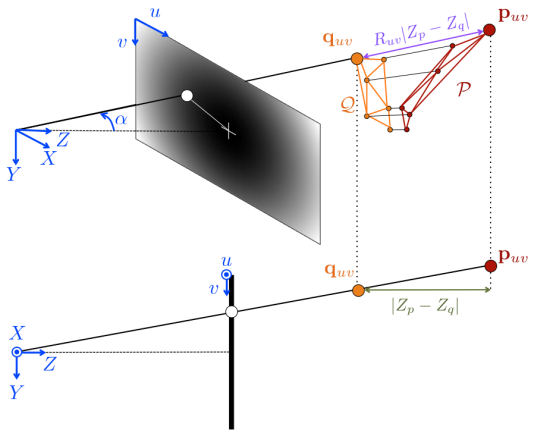


Fig. 2: Geometric interpretation of R_{uv}. Perspective (top) and lateral (bottom) view of the pinhole projection of point clouds $\mathcal{Q}$ (orange) and $\mathcal{P}$ (red). We show (for the white pixel) that the depth offset $|Z_p - Z_q|$ (green) differs from the point distance (purple) by scalar factor R_{uv}. This factor is constant on ellipses centred at the camera principal point, and increases with the sizes of the ellipses. In the top figure, we outline these elipses, which characterize image $\mathbf{R}$. We also show (in blue) the camera and image frames, as well as α, the angle between the pixel projecting ray and the optical axis.

produce suboptimal results on many problems, and longer term planning should be preferred. Yet, since the focus of this paper is not planning, we have posed this assumption to simplify the point cloud shaping implementation. Below, we explain how we have defined d.

B. Distance between two Visible Point Clouds

Consider two point clouds $\mathcal{P}$ and $\mathcal{Q}$, visible by the depth camera in images $\mathbf{L}_P$ and $\mathbf{L}_Q$. In this Section, we define the metric $d(\mathcal{P}, \mathcal{Q}) \in \mathcal{R}^+$ used in our work to measure the distance between $\mathcal{P}$ and $\mathcal{Q}$.

A common way of defining d is to set it equal to the mean of the distances between all the pairs $(\mathcal{P}, \mathcal{Q})$ of points matched in the two point clouds:

$$d(\mathcal{P}, \mathcal{Q}) = \frac{1}{\dim \mathcal{M}} \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{M}} \|\mathbf{p} - \mathbf{q}\|_2 . \tag{16}$$

Alternative approaches are the Chamfer distance or the Earth Mover's Distance [27]. However, all these metrics require computing $\mathcal{M}$, the set denoting the association between the point clouds, which is defined as:

$$\mathcal{M} = \left\{ (\mathbf{p}_l, \mathbf{q}_l) : \forall \mathbf{p}_l \in \mathcal{P}, \mathbf{q}_l = \underset{m}{\operatorname{argmin}} \|\mathbf{p}_l - \mathbf{q}_m\|_2 \in \mathcal{Q} \right\}. \tag{17}$$

This matching operation can be solved by the well known point-to-point ICP algorithm [15]. Although it is one of the widely used algorithms for aligning three dimensional models given an initial guess of the required rigid transformation, ICP's complexity (quadratic with the point cloud dimension) makes it very slow.

In this pilot study, the task's complexity is reduced, since the robot can realize a limited number of actions. For human-like molding – which requires many more action types and positions – the distance computation time becomes crucial, at both learning and planning steps. In this perspective, a metric based on depth images will scale up much better than one based on point clouds. Furthermore, a point cloud uses approximately %60 more memory than the equivalent depth image. Hence, using point clouds will substantially increase memory usage and may not be suitable for deep learning on consumer GPUs.

For all these reasons, we propose to match points from $\mathcal{P}$ and $\mathcal{Q}$ which project to the same pixel in images $\mathbf{L}_P$ and $\mathbf{L}_Q$ (see Fig. 2). Naming

$$\begin{cases} \mathbf{p}_{uv} = \mathbf{m}(u, v, L^p_{uv}) \\ \mathbf{q}_{uv} = \mathbf{m}(u, v, L^q_{uv}) \end{cases} \tag{18}$$

with L^p_{uv} (respectively, L^q_{uv}) the luminance of pixel (u, v) in depth image $\mathbf{L}_P$ ($\mathbf{L}_Q$), we propose to use:

$$\mathcal{M} = \{\mathbf{p}_{uv}, \mathbf{q}_{uv}, \forall (u, v) \in \mathcal{I}\}. \tag{19}$$

Note that since their pixel projection (u, v) is the same, $\mathbf{p}_{uv}$ and $\mathbf{q}_{uv}$ have identical x and y. Applying first (11) and then (3), we can express their relative distance as:

$$\begin{aligned} \|\mathbf{p} - \mathbf{q}\|_2 &= \sqrt{(xZ_p - xZ_q)^2 + (yZ_p - yZ_q)^2 + (Z_p - Z_q)^2} \\ &= \sqrt{1 + x^2 + y^2}\,|Z_p - Z_q| \\ &= R_{uv} \frac{Z_{max} - Z_{min}}{2^b - 1} |L^p_{uv} - L^q_{uv}|. \end{aligned} \tag{20}$$

In this equation we have noted: $Z_p = Z(L^p_{uv})$, $Z_q = Z(L^q_{uv})$ and $R_{uv} = \sqrt{1 + \left(\frac{u-u_0}{f_x}\right)^2 + \left(\frac{v-v_0}{f_y}\right)^2}$. The value R_{uv} is independent from the point clouds, is constant on ellipses centered at the camera principal point (where it is 1), and it increases with the sizes of the ellipses. A geometric interpretation when $f_x = f_y$ is $R_{uv} = 1/cos\alpha$, with α the angle between the pixel projecting ray and the optical axis (see Fig. 2). For the Intel RealSense D435, with nominal parameters[3] $u_0 = 424$, $v_0 = 240$, $f_x = 415$, $f_y = 373$, R_{uv} varies between $\underline{R} = 1$ (for the principal point) and $\overline{R} = 1.568$ (for the four corner pixels).

Injecting (20) in (16) for all the pixels yields:

$$d(\mathcal{P}, \mathcal{Q}) = \frac{Z_{max} - Z_{min}}{wh(2^b - 1)} \sum_{(u,v)\in\mathcal{I}} R_{uv} |L^p_{uv} - L^q_{uv}|. \tag{21}$$

Since Z_{max} and Z_{min} are the same for all pairs of images $(\mathbf{L}_P, \mathbf{L}_Q)$, we can remove them from the equation. We can also divide by $\overline{R}$, to redefine the distance on unit interval $\mathcal{U}$:

$$d(\mathcal{P}, \mathcal{Q}) = \frac{\sum_{(u,v)\in\mathcal{I}} R_{uv} |L^p_{uv} - L^q_{uv}|}{wh(2^b - 1)\overline{R}} \in \mathcal{U} = [0, 1]. \tag{22}$$

Let us now define matrix

$$\mathbf{R} = \frac{1}{(2^b - 1)\overline{R}} \begin{pmatrix} R_{11} & \cdots & R_{1w} \\ \vdots & \ddots & \vdots \\ R_{h1} & \cdots & R_{hw} \end{pmatrix} \tag{23}$$

[3]See https://www.intelrealsense.com/depth-camera-d435

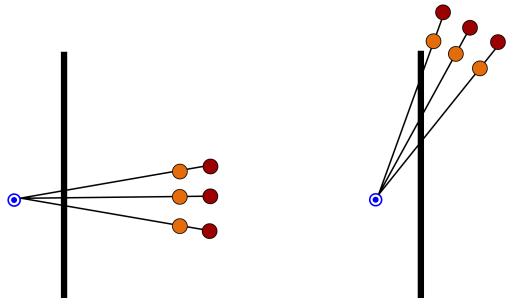


Fig. 3: The same two point clouds, red and orange, each composed of three points, seen with different camera orientations (left) and (right).

shown in Fig. 2 for the Intel RealSense D435. Then, naming

$$\mathbf{I} = \mathbf{R} \circ \mathbf{L} \in \mathcal{U}^{w \times h}, \tag{24}$$

with $\circ$ the Hadamard product, d can be expressed as a function of these new normalized images $\mathbf{I}$:

$$d(\mathcal{P}, \mathcal{Q}) = d(\mathbf{I}_p, \mathbf{I}_q) = \frac{1}{wh} \sum_{(u,v) \in \mathcal{I}} |I^p_{uv} - I^q_{uv}|. \tag{25}$$

Therefore, in the rest of the paper (except when specified) we work with images $\mathbf{I}$, rather than $\mathbf{L}$.

We have analytically proved that our distance is equivalent to the weighted pixelwise difference between the two depth images. To illustrate the difference between our distance and an unweighted version, refer to the example of Fig. 3. The same two point clouds, red and orange, each composed of three points, are seen with different camera orientations in Fig. 3(left) and Fig. 3(right). With our technique, the distance between the point clouds is the same in Fig. 3(left) and Fig. 3(right), as it should be (i.e., viewpoint invariant). Instead, with the unweighted version, which merely uses the depth differences, the distance in Fig. 3(right) will be much smaller than in Fig. 3(left). To summarize, our method calculates the geometric distance between two points instead of its projection along the camera axis (i.e., instead of the difference between the two points' depths). Hence, we think that our metric is faster and at least as easy to interpret than other similar metrics, such as Chamfer distance or the Earth Mover's Distance. Thus, it can also be useful for planning algorithms.

Proposition 1. *If both $\mathcal{P}$ and $\mathcal{Q}$ are visible point clouds, $d(\mathcal{P}, \mathcal{Q}) = 0$ if and only if $\mathcal{P} = \mathcal{Q}$. Therefore, d is an appropriate metric for the Point Cloud Shaping Task.*

Proof: Since $d(\mathcal{P}, \mathcal{Q})$ is a sum of absolute values, if it is null, $I^p_{uv} = I^q_{uv} \forall (u, v) \in \mathcal{I}$. Since each I^p_{uv} (respectively, I^q_{uv}) is proportional to L^p_{uv} (respectively, L^q_{uv}) by the corresponding element of $\mathbf{R}$, if $d(\mathcal{P}, \mathcal{Q}) = 0$ then $L^p_{uv} = L^q_{uv}$. Then, mapping the two images with $\mathbf{m}$ will yield identical point clouds, so if $d(\mathcal{P}, \mathcal{Q}) = 0$ then $\mathcal{P} = \mathcal{Q}$. Conversely: if the point clouds are identical ($\mathcal{P} = \mathcal{Q}$), applying $\mathbf{m}^{-1}$ to each one will yield identical depth images ($\mathbf{L}_p = \mathbf{L}_q$). Then, applying (24) followed by (25) leads to $d(\mathcal{P}, \mathcal{Q}) = 0$. ■

IV. Predicting the Action Effect on a Point Cloud

In this section, we explain how to predict the effect of an action on a point cloud.

TABLE II: Pixel coordinates of the top left $(\underline{u}, \underline{v})$ and bottom right $(\overline{u}, \overline{v})$ pixel of the rectangle embedding the image of $\mathcal{B}$.

X^-	$\underline{u}$
≥ 0	$\max\left(1, \min\left(\lfloor f_x X^- / Z^+ + u_0 \rceil, w\right)\right)$
< 0	$\max\left(1, \min\left(\lfloor f_x X^- / Z^- + u_0 \rceil, w\right)\right)$
Y^-	$\underline{v}$
≥ 0	$\max\left(1, \min\left(\lfloor f_y Y^- / Z^+ + v_0 \rceil, h\right)\right)$
< 0	$\max\left(1, \min\left(\lfloor f_y Y^- / Z^- + v_0 \rceil, h\right)\right)$
X^+	$\overline{u}$
≥ 0	$\max\left(1, \min\left(\lfloor f_x X^+ / Z^- + u_0 \rceil, w\right)\right)$
< 0	$\max\left(1, \min\left(\lfloor f_x X^+ / Z^+ + u_0 \rceil, w\right)\right)$
Y^+	$\overline{v}$
≥ 0	$\max\left(1, \min\left(\lfloor f_y Y^+ / Z^- + v_0 \rceil, h\right)\right)$
< 0	$\max\left(1, \min\left(\lfloor f_y Y^+ / Z^+ + v_0 \rceil, h\right)\right)$

A. Outline

At each iteration, action $\mathbf{a}$ changes the current point cloud $\mathcal{P}$, to produce a new point cloud $\mathcal{P}_\mathbf{a}$, according to (13). Since $\mathbf{a}$ can be any action within $\mathcal{A} = \{\mathbf{a}^{ij}\}$, we must rewrite (13) for each possible $\mathbf{a}^{ij}$. We can do this by embedding the effect of $\mathbf{a}^{ij}$ within the expression of $\mathbf{f}$:

$$\mathcal{P}_{\mathbf{a}^{ij}} = \mathbf{f}^{ij}(\mathcal{P}), \tag{26}$$

with:

$$\mathbf{f}^{ij}: \quad \mathcal{R}^{3 \times w \times h} \to \mathcal{R}^{3 \times w \times h}. \tag{27}$$

To predict the effect of *each* possible action $\mathbf{a}^{ij}$, we need to know each $\mathbf{f}^{ij}$. Both the domain and codomain of these functions have very high dimension ($3 \times w \times h$), and we need $\dim \mathcal{A}$ of such functions! This clearly makes the prediction problem intractable, unless we exploit the other work assumptions to simplify it.

B. Simplifications

First, since both the prior and posterior point clouds are visible, we can project them on the corresponding images, by applying (6) followed by (24). Then, we can rewrite (27) as:

$$\begin{aligned} \mathbf{f}^{ij}: \quad & \mathcal{U}^{w \times h} \to \mathcal{U}^{w \times h} \\ & \mathbf{I} \mapsto \mathbf{I}_{\mathbf{a}^{ij}}. \end{aligned} \tag{28}$$

By operating on images instead of point clouds, we have reduced the dimensions of the functions' domain and codomain from $3 \times w \times h$ to $w \times h$.

We can further simplify the problem, by considering Hypothesis 2: the effect of actions is bounded. The projection of box $\mathcal{B}_{ij}$ in the depth image can be embedded within a rectangular region of interest (ROI). Table II gives the coordinates of the top left $(\underline{u}, \underline{v})$ and bottom right $(\overline{u}, \overline{v})$ pixel of this ROI, depending on the signs of $X^\pm$ and $Y^\pm$ ($Z^\pm > 0$ for all points of a visible point cloud). We derived the pixel coordinates in the second column of the table by applying (6), rounding to the closest integer (with $\lfloor \rceil$) and then clamping the result in the field of view. To illustrate this result, we show in Fig. 4 two examples: a red bounding box with all four values (X^+, X^-, Y^+, Y^-) negative and a green bounding box where all four values are positive.

Note that the clamping can be removed, since under Hypothesis 2, all $\mathcal{B}_{ij}$ are entirely visible. This hypothesis

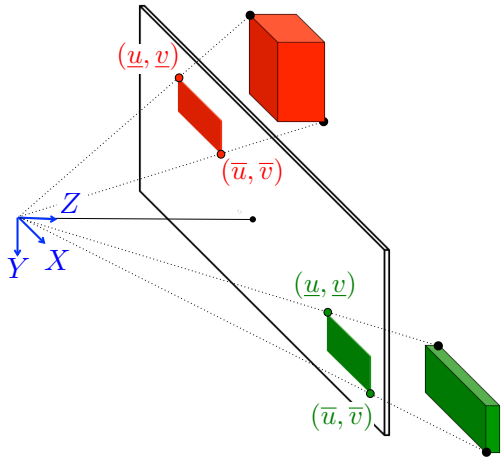


Fig. 4: Two examples of bounding boxes (a red one with all four values X^+, X^-, Y^+, Y^- negative and a green one, where all four values are positive) with the corresponding rectangular regions of interest (ROI) in the image plane. The ROI are characterized by the coordinates of the top left $(\underline{u},\underline{v})$ and bottom right $(\overline{u},\overline{v})$ pixel. Note that these pixels are the projections of different corners of the two bounding boxes.

(equivalent to $\underline{u},\overline{u} \in [1,w]$ and $\underline{v},\overline{v} \in [1,h]$) can be guaranteed by setting the action positions $\mathbf{t}$ such that:

$$\begin{cases} \frac{(1-u_0)Z^+}{f_x} + \Delta X \leq X \leq \frac{(w-u_0)Z^-}{f_x} - \Delta X \\ \frac{(1-v_0)Z^+}{f_y} + \Delta Y \leq Y \leq \frac{(h-v_0)Z^-}{f_y} - \Delta Y \end{cases} . \tag{29}$$

Under Hypothesis 3 (i.e., $Z^- \approx Z^+ \approx Z$) the ROI top left and bottom right pixels are always:

$$\begin{cases} (\underline{u},\underline{v}) = (f_x X^-/Z + u_0, f_y Y^-/Z + v_0) \\ (\overline{u},\overline{v}) = (f_x X^+/Z + u_0, f_y Y^+/Z + v_0) \end{cases} . \tag{30}$$

Note that since all actions have the same Z, the ROI width and height, $\underline{w} = \overline{u} - \underline{u} = 2f_x\Delta X/Z$ and $\underline{h} = \overline{v} - \underline{v} = 2f_y\Delta Y/Z$, are unique for a given $\mathbf{a}^i$ (i.e., they do not depend on $\mathbf{t}$).

Let us denote

$$\underline{\mathbf{I}} = \begin{pmatrix} I_{\underline{u}\,\underline{v}} & \cdots & I_{\overline{u}\,\underline{v}} \\ \vdots & \ddots & \vdots \\ I_{\underline{u}\,\overline{v}} & \cdots & I_{\overline{u}\,\overline{v}} \end{pmatrix} \in \mathcal{U}^{\underline{w}\times\underline{h}} \tag{31}$$

the submatrix of $\mathbf{I}$, containing the values of the pixels which could change after an action in $\mathcal{B}_{ij}$. Action $\mathbf{a}^{ij}$ will transform $\underline{\mathbf{I}}$ into a new image:

$$\underline{\mathbf{I}}_{\mathbf{a}^{ij}} = \begin{pmatrix} I^{ij}_{\underline{u}\,\underline{v}} & \cdots & I^{ij}_{\overline{u}\,\underline{v}} \\ \vdots & \ddots & \vdots \\ I^{ij}_{\underline{u}\,\overline{v}} & \cdots & I^{ij}_{\overline{u}\,\overline{v}} \end{pmatrix} \in \mathcal{U}^{\underline{w}\times\underline{h}}. \tag{32}$$

By focusing on a region of the image, we have further reduced the dimensions of the domain and codomain, from $w \times h$ to $\underline{w} \times \underline{h}$:

$$\begin{aligned} \mathbf{f}^{ij}: \quad & \mathcal{U}^{\underline{w}\times\underline{h}} \to \mathcal{U}^{\underline{w}\times\underline{h}} \\ & \underline{\mathbf{I}} \mapsto \underline{\mathbf{I}}_{\mathbf{a}^{ij}}. \end{aligned} \tag{33}$$

Yet, we still need to know the function $\mathbf{f}^{ij}$ for each combination of action i and position j.

To further simplify the problem, we exploit Hypothesis 4: actions are position invariant. This last simplification reduces the number of functions required: instead of $\dim \mathcal{A}$ functions, it is sufficient to know the function of each $\mathbf{a}^i$:

$$\begin{aligned} \mathbf{f}^{i}: \quad & \mathcal{U}^{\underline{w}\times\underline{h}} \to \mathcal{U}^{\underline{w}\times\underline{h}} \\ & \underline{\mathbf{I}} \mapsto \underline{\mathbf{I}}_{\mathbf{a}^{i}}. \end{aligned} \tag{34}$$

In the next subsection, we explain how we determine the function $\mathbf{f}^i$ predicting the effect of action $\mathbf{a}^i$ on a region of interest $\underline{\mathbf{I}}$.

C. Predicting the action effect on a region of interest

To predict the effect of $\mathbf{a}^i$ (noted $\mathbf{a}$ for simplicity) on $\underline{\mathbf{I}}$, we have tested three different functions $\mathbf{f}$. To design them, we exploit – for each $\mathbf{a}$ – a prerecorded dataset of pairs of regions of interest prior/posterior to the action:

$$\mathcal{G} = \left\{ (\underline{\mathbf{I}}, \underline{\mathbf{I}}_{\mathbf{a}})_1, \ldots, (\underline{\mathbf{I}}, \underline{\mathbf{I}}_{\mathbf{a}})_{\dim \mathcal{G}} \right\} \tag{35}$$

The three designs are: *difference compensation*, *refined generator*, and *difference compensation with refined generator* (denoted respectively $\mathbf{f}_{\mathrm{D}}$, $\mathbf{f}_{\mathrm{CR}}$, and $\mathbf{f}_{\mathrm{DCR}}$). The designs are outlined in Fig. 5; we will detail them in this section, after having explained how to build the dataset (35) of prior/posterior pairs.

1) Building a dataset of prior/posterior image ROI: We record a sequence of depth images $\mathbf{L}$, while repeatedly applying action $\mathbf{a}$. We use (24) to convert each image $\mathbf{L}$ into the corresponding $\mathbf{I}$.

We remove all images where the environment is occluded (typically, during the action), and extract pairs of prior/posterior images, which represent the image before/after the action. Then, we manually annotate, in these images, the ROI which contains a "visible" (by human standards) difference in depth[4]. Afterwards, we adjust the width $\underline{w}$ and height $\underline{h}$ of all the ROI to the mean of all the annotated ones, and we fix their positions so that they are identical in both images of the pair. These ROI will constitute (35).

2) Difference compensation: A first simple image processing operation consists in approximating the effect of $\mathbf{a}$ on $\underline{\mathbf{I}}$, by the average (over the dataset (35)) of the ROI differences

$$\Delta\underline{\mathbf{I}} = \frac{1}{\dim \mathcal{G}} \sum^{\mathcal{G}} (\underline{\mathbf{I}}_{\mathbf{a}} - \underline{\mathbf{I}}) . \tag{36}$$

This image average is added to $\underline{\mathbf{I}}$, to obtain a rough prediction of the ROI, after having applied $\mathbf{a}$:

$$\begin{aligned} \mathbf{f}_{\mathrm{D}}: \quad & \mathcal{U}^{\underline{w}\times\underline{h}} \to \mathcal{U}^{\underline{w}\times\underline{h}} \\ & \underline{\mathbf{I}} \mapsto \underline{\mathbf{I}} + \Delta\underline{\mathbf{I}}. \end{aligned} \tag{37}$$

Although this operation may lead some pixels to saturate (to 0 or 1), this occurs very rarely in practice.

3) Refined Generator: Our generator is a type of artificial neural network which is constituted by two main parts. The encoder maps the input into feature representations at multiple levels and the decoder maps the features onto the pixel space. We design and train (with the prior/posterior images from $\mathcal{G}$) a generator to generate a ROI from an input one:

$$\begin{aligned} \mathbf{f}_{\mathrm{C}}: \quad & \mathcal{U}^{\underline{w}\times\underline{h}} \to \mathcal{U}^{\underline{w}\times\underline{h}} \\ & \underline{\mathbf{I}} \mapsto \mathbf{f}_{\mathrm{C}}(\underline{\mathbf{I}}) . \end{aligned} \tag{38}$$

[4]The annotation could be automated using standard image processing techniques, similar to the ones we developed in [5].

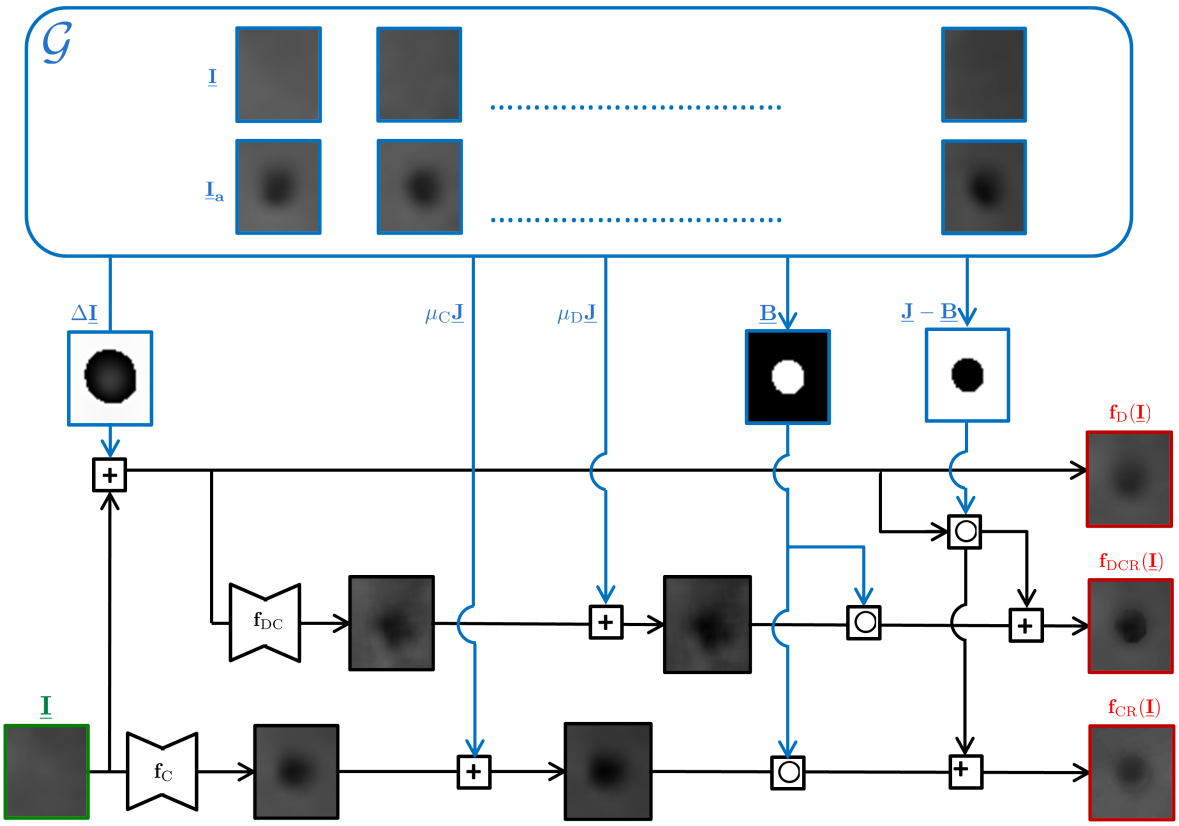


Fig. 5: The three prediction functions $\mathbf{f}_\mathrm{D}$, $\mathbf{f}_\mathrm{DCR}$, and $\mathbf{f}_\mathrm{CR}$, illustrated (in red) for the poke action. For the purpose of illustration, in the image $\Delta\underline{\mathbf{I}}$ we have set all negative pixels to white (i.e., to unitary value).

We have implemented $\mathbf{f}_\mathrm{C}$ as a modified U-net image generator [28], similar to the one used by the pix2pix generative adversarial network [29], with the following specifications (illustrated in Fig. 6). For the encoder, the size of the input layer varies according to the ROI dimensions $(\underline{w}, \underline{h})$. The input is followed by 4 downscale layers, each consisting of two 3x3 convolution layers followed by a 2x2 maxpooling layer. Filter sizes of the convolution layers are 32, 64, 128 and 256, respectively. The decoder has 5 upscale layers and 1 output layer. Upscale layers consist of two 3x3 convolution layers, followed by a 2x2 upsampling layer. Filter size of convolution layers are 512, 256, 128, 64 and 32, respectively. Outputs of the second convolution layer from downscale layers are concatenated with the output of the symmetrical upsampling layer, before being fed to the following upscale layers. The output layer is a 1x1 convolution layer with $\tanh$ activation function. All the strides are set to 1 and rectified linear unit activation is used in all the convolution layers, except for the output layer. We use as loss function between the output of $\mathbf{f}_\mathrm{C}$ and the corresponding posterior $\underline{\mathbf{I}}_\mathbf{a}$ (ground truth or expected output, taken from $\mathcal{G}$), the distance d defined in (25):

$$d\left(\underline{\mathbf{I}}_\mathbf{a}, \mathbf{f}_\mathrm{C}\left(\underline{\mathbf{I}}\right)\right) = \left|\underline{\mathbf{I}}_\mathbf{a} - \mathbf{f}_\mathrm{C}\left(\underline{\mathbf{I}}\right)\right|. \tag{39}$$

Note that this loss function resembles the mean absolute error, commonly used in Convolutional Neural Networks[5], with the addition of the $R_{uv}/\overline{R}$ weights.

[5]See for instance the *MeanAbsoluteError* class in *Keras*: https://keras.io/api/losses/regression_losses

We have realized experimentally that $\mathbf{f}_\mathrm{C}$ is not very accurate in predicting changes in the areas of $\underline{\mathbf{I}}$ the least affected by the action. In the areas the most affected by the action, the output of $\mathbf{f}_\mathrm{C}$ is accurate, apart from a slight difference in its mean pixel value. To solve both issues, we have added a *refining step* (described below) to the output of the generator $\mathbf{f}_\mathrm{C}$.

We start by building a boolean matrix $\underline{\mathbf{B}}$ by applying Otsu thresholding [30] followed by morphological erosion, to the dataset average image difference, $\Delta\underline{\mathbf{I}}$ defined in (36). Then:

- pixels which are the least modified by the action are null in $\underline{\mathbf{B}}$; we set their values to those of the corresponding pixels in image $\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}}$.
- pixels which are the most modified by the action are unitary in $\underline{\mathbf{B}}$; we shift their values, output by $\mathbf{f}_\mathrm{C}$, by an offset μ_C, such that their mean value is identical to that of the same pixels in $\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}}$.

In summary:

$$\begin{aligned} \mathbf{f}_\mathrm{CR}: \quad & \mathcal{U}^{\underline{w}\times\underline{h}} \to \mathcal{U}^{\underline{w}\times\underline{h}} \\ & \underline{\mathbf{I}} \mapsto \underline{\mathbf{B}} \circ \left(\mathbf{f}_\mathrm{C}\left(\underline{\mathbf{I}}\right) + \mu_\mathrm{C}\underline{\mathbf{J}}\right) + \\ & \quad + \left(\underline{\mathbf{J}} - \underline{\mathbf{B}}\right) \circ \left(\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}}\right), \end{aligned} \tag{40}$$

with $\underline{\mathbf{J}}$ a $(\underline{w}$ x $\underline{h})$ matrix of ones, and $\mu_\mathrm{C} \in [-1, 1]$ the mean of the non-zero pixels of $\underline{\mathbf{B}} \circ (\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}} - \mathbf{f}_\mathrm{C}(\underline{\mathbf{I}}))$.

4) Difference compensation with Refined Generator: The difference compensation function $\mathbf{f}_\mathrm{D}$ has the advantage of not requiring a training process. A drawback is that adding an image average introduces noise (blur) on $\underline{\mathbf{I}}$. The authors of [31] have shown that to denoise images, convolutional

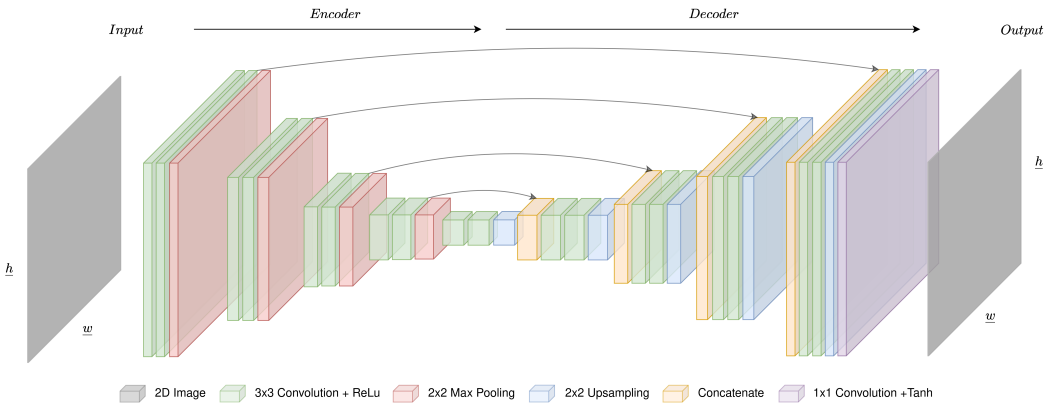


Fig. 6: Architecture of the Generator Networks $\mathbf{f}_{\mathrm{C}}$ and $\mathbf{f}_{\mathrm{DC}}$.

autoencoders, which are simpler architectures than our generator, outperform wavelets and Markov random fields on small samples of training images. Therefore, our third approach consists in passing $\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}}$ to a new *denoising* generator, denoted $\mathbf{f}_{\mathrm{DC}}$, and then refining the output, as in (40). We have designed $\mathbf{f}_{\mathrm{DC}}$ with the same architecture as $\mathbf{f}_{\mathrm{C}}$. Yet, the decoder in $\mathbf{f}_{\mathrm{DC}}$ should map the latent space to a denoised version of its input. Therefore, we train this network with the prior images of $\mathcal{G}$ with $\Delta\underline{\mathbf{I}}$ added to them. The loss function of $\mathbf{f}_{\mathrm{DC}}$ is the distance between its output and the corresponding posterior ROI, $\underline{\mathbf{I}}_{\mathbf{a}}$ (from $\mathcal{G}$):

$$d\left(\underline{\mathbf{I}}_{\mathbf{a}}, \mathbf{f}_{\mathrm{DC}}\left(\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}}\right)\right) = \left|\underline{\mathbf{I}}_{\mathbf{a}} - \mathbf{f}_{\mathrm{DC}}\left(\underline{\mathbf{Ia}} + \Delta\underline{\mathbf{I}}\right)\right|. \tag{41}$$

The images output by $\mathbf{f}_{\mathrm{DC}}$ have the same issues seen for $\mathbf{f}_{\mathrm{C}}$. Therefore, we apply a similar refinement step with boolean matrix $\underline{\mathbf{B}}$, and design $\mathbf{f}_{\mathrm{DCR}}$ as:

$$\begin{aligned} \mathbf{f}_{\mathrm{DCR}}: \quad & \mathcal{U}^{\underline{w}\times\underline{h}} \to \mathcal{U}^{\underline{w}\times\underline{h}} \\ & \underline{\mathbf{I}} \mapsto \underline{\mathbf{B}} \circ \left(\mathbf{f}_{\mathrm{DC}}\left(\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}}\right) + \mu_{\mathrm{D}}\underline{\mathbf{J}}\right) + \\ & \qquad + \left(\underline{\mathbf{J}} - \underline{\mathbf{B}}\right) \circ \left(\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}}\right), \end{aligned} \tag{42}$$

with $\mu_{\mathrm{D}} \in [-1, 1]$ the mean of the non-zero pixels of $\underline{\mathbf{B}} \circ \left(\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}} - \mathbf{f}_{\mathrm{DC}}\left(\underline{\mathbf{I}} + \Delta\underline{\mathbf{I}}\right)\right)$.

We have presented three alternative functions $\mathbf{f}$ for predicting the effect of an action on a region of interest $\underline{\mathbf{I}}$: $\mathbf{f}_{\mathrm{D}}$, $\mathbf{f}_{\mathrm{CR}}$, and $\mathbf{f}_{\mathrm{DCR}}$. Having chosen one of the three (either (37), (40) or (42)), we can replace the resulting (*predicted*) submatrix in the original image $\mathbf{I}$, to obtain:

$$\mathbf{I}_{\mathbf{a}^{ij}} = \begin{pmatrix} I_{11} & \cdots & \cdots & \cdots & \cdots & \cdots & I_{1w} \\ \cdots & \cdots & I^i_{\underline{u}\,\underline{v}} & \cdots & I^i_{\overline{u}\underline{v}} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \cdots & \cdots & I^i_{\underline{u}\,\overline{v}} & \cdots & I^i_{\overline{u}\,\overline{v}} & \cdots & \cdots \\ I_{h1} & \cdots & \cdots & \cdots & \cdots & \cdots & I_{hw} \end{pmatrix}. \tag{43}$$

Note that we have removed from the second term the j dependency (pedix), which appeared in (32) and which is irrelevant since actions are position invariant.

V. Finding the Best Sequence of Actions for Point Cloud Shaping

In this section, we explain how to compute the sequence of actions $\mathbf{s}_* = \{\mathbf{a}_1, \ldots, \mathbf{a}_K\}$ for *point cloud shaping*, i.e., for modifying the environment from initial point cloud $\mathcal{P}_0$ to final desired point cloud $\mathcal{P}_*$.

By mapping point clouds to images with (6), we can instantiate this path planning problem in image space, i.e., find $\mathbf{s}$ which modifies the environment from $\mathbf{L}_0$ to $\mathbf{L}_*$ (i.e., from the depth image of $\mathcal{P}_0$ to that of $\mathcal{P}_*$). Furthermore, to express distance d according to (25), we will work with weighed normalized images $\mathbf{I}_0$ and $\mathbf{I}_*$ instead of $\mathbf{L}_0$ and $\mathbf{L}_*$ by applying (24). Within a set of candidate sequences of actions $\mathcal{S}$ each leading from $\mathbf{I}_0$ to a different image $\mathbf{I}_{|\mathbf{s}}$, we will choose the one leading to the image that is the closest to $\mathbf{I}_*$:

$$\mathbf{s}_* = \underset{\mathbf{s}\in\mathcal{S}}{\operatorname{argmin}} \; d\left(\mathbf{I}_{|\mathbf{s}}, \mathbf{I}_*\right). \tag{44}$$

This sequence corresponds to the shortest path from $\mathbf{I}_0$ to $\mathbf{I}_*$, in the tree (shown in Fig. 7 with an example of forward search) of all possible images generated by all possible actions in $\mathcal{A}$. Finding the shortest path in a tree is a classic motion planning problem, which can be solved using many state of art algorithms (A*, RRT, etc) out of scope here. We leave to path planning experts the pleasure of proposing the best among such algorithms. Since the focus of our paper is *on how to build the tree of posterior images*, we propose the following forward search algorithm (refer to **Algorithm 1**):

1) at each step k, generate a tree from $\mathbf{I}_k$ to all possible (i.e., obtained after executing any action in $\mathcal{A}$) posterior images $\mathbf{I}_{k+1}$;
2) update the sequence of actions with the action that yields the image (among all $\mathbf{I}_{k+1}$) which is the closest to $\mathbf{I}_*$ and set this image as the $\mathbf{I}_k$ for the following step;
3) loop until none of the posterior images $\mathbf{I}_{k+1}$ is closer to $\mathbf{I}_*$ than $\mathbf{I}_k$ – in other words, loop while (15) is verified;
4) output the action sequence $\mathbf{s}_*$.

The most challenging part of the algorithm is the prediction of the effect of action $\mathbf{a}^i$ on the image (lines 6 to 8). This requires an appropriate choice of $\mathbf{f}$, among the three implementations proposed in Sec. IV.

A. Reinsertion of the ROI

Interested readers may find more advanced methods (often based on motion planning or on machine learning) in the literature on automatic action selection. A planning method for finding intermediate states to shape a flexible wire is presented

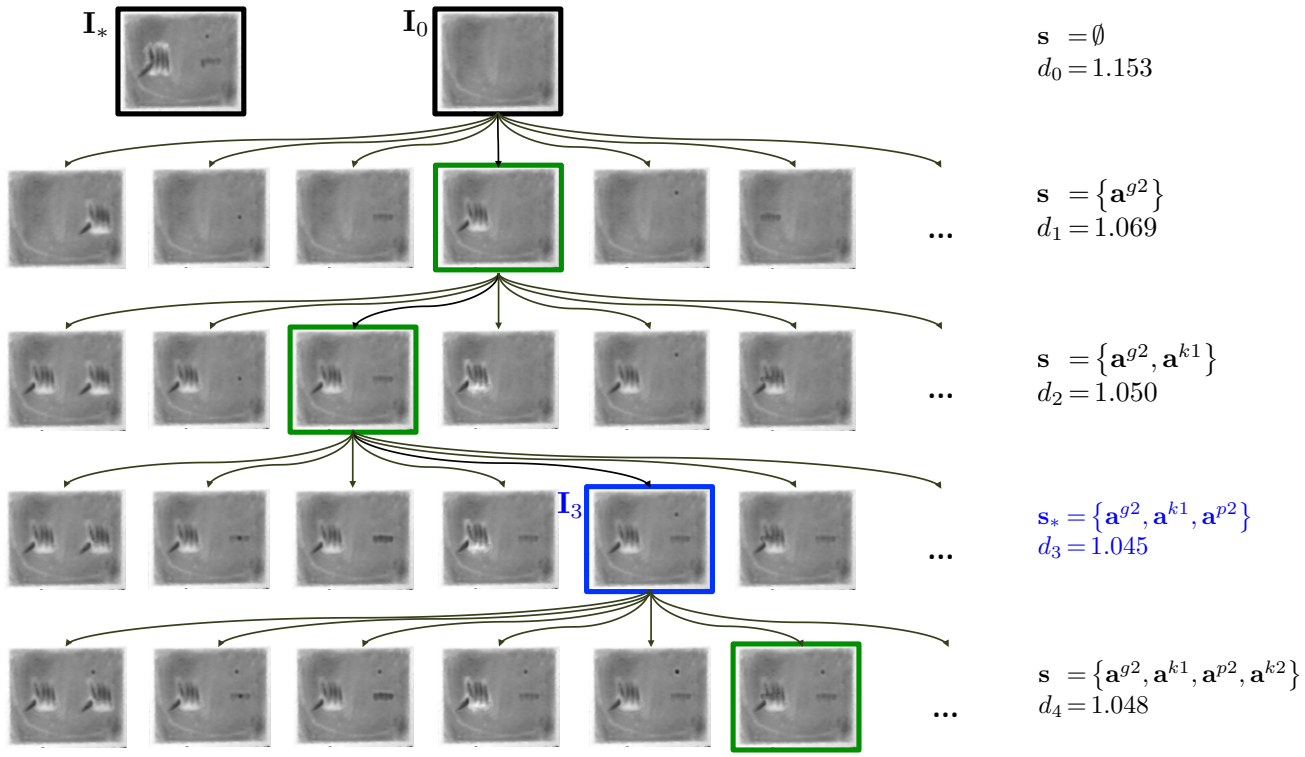


Fig. 7: Forward search in the tree of posterior images, based on **Algorithm 1**. Given initial image $\mathbf{I}_0$ and desired image $\mathbf{I}_*$, at each step the sequence $\mathbf{s}$ is updated with the action which yields the image (green) that is the closest (has smallest distance d) to $\mathbf{I}_*$. In this example, the search stops after 3 steps with the sequence $\mathbf{s}_*$ in blue, since any fourth action would increase the distance ($d_4 > d_3$). Note the similarity between the resulting image ($\mathbf{I}_3$ in blue) and $\mathbf{I}_*$. For the sake of clarity, at each step we only show 6 of the total $\dim \mathcal{A}$ posterior images. Distances are expressed in mm.

Algorithm 1 Action sequence planner

Input: Initial image $\mathbf{L}_0$ and final desired image $\mathbf{L}_*$.
Output: Sequence of actions $\mathbf{s}_* = \{\mathbf{a}_1, \ldots, \mathbf{a}_K\}$.

initialize $k = 0$, $\mathbf{s} = \emptyset$
apply (24) to obtain $\mathbf{I}_{0|\emptyset}$ from $\mathbf{L}_0$ and $\mathbf{I}_*$ from $\mathbf{L}_*$
compute $d_0 = d\left(\mathbf{I}_{0|\emptyset}, \mathbf{I}_*\right)$ with (25)
loop
  for each action $\mathbf{a}^{ij} \in \mathcal{A}$ **do**
    extract submatrix $\underline{\mathbf{I}}$ from $\mathbf{I}_{k|\mathbf{s}}$ via (31)
    using (34) predict effect of $\mathbf{a}^{ij}$ on $\underline{\mathbf{I}}$: $\underline{\mathbf{I}}_{\mathbf{a}^{ij}}$
    with (43) inject $\underline{\mathbf{I}}_{\mathbf{a}^{ij}}$ in $\mathbf{I}_{k|\mathbf{s}}$ to get $\mathbf{I}_{k+1|\{\mathbf{s},\mathbf{a}^{ij}\}}$
    compute $d_{ij} = d\left(\mathbf{I}_{k+1|\{\mathbf{s},\mathbf{a}^{ij}\}}, \mathbf{I}_*\right)$ with (25)
  end for
  if $\exists\, d_{ij} < d_k$ **then**
    find best action $\mathbf{a}_k = \underset{\mathbf{a}^{ij} \in \mathcal{A}}{\operatorname{argmin}}\ d_{ij}$
    update sequence $\mathbf{s} = \{\mathbf{s}, \mathbf{a}_k\}$
    $k = k + 1$
    $d_k = d\left(\mathbf{I}_{k|\mathbf{s}}, \mathbf{I}_*\right)$
  else
    return $\mathbf{s}_* = \mathbf{s}$
  end if
end loop

in [32]. The authors of [33] use deep reinforcement learning to choose within a finite number of actions, whereas [34] relies on imitation learning to choose actions for cloth manipulating.

VI. Experiments

In this section, we present the experiments that we did to assess our framework. The software and a video of the experiments are available online at: https://github.com/gursoyege/lirmm-farine. The video is also attached to the paper and available at https://youtu.be/fLGf7c3cbKk.

A. Setup

Figure 8 shows our experimental setup. We focus on flour manipulation, with the three actions grasp, knock and poke shown – along with their parameters – in Table I. We did not apply actions push and pinch, since the robot actuation restrains it from realizing these actions. The three actions can be applied in $N = 15$ positions; hence, for these experiments, the total number of actions is $\dim \mathcal{A} = 3N = 45$. Flour is placed within a custom-made wooden box of size 400 x 500 mm. We use the right arm (KUKA LWR 4+) and Shadow Dexterous Hand of the BAZAR robot [35] with an Intel Realsense D435 depth camera rigidly linked to BAZAR's right wrist. The camera parameters obtained after calibration are $u_0 = 421$, $v_0 = 243$, $f_x = 433$, $f_y = 433$. Although the image resolution is 848×480, to calculate d we have focused on the region of interest containing the box and flour, which has size 480×395.

The actions are realized using the RKCL framework[6]. This C++ library allows to quickly set up a kinematic controller for

[6] https://gite.lirmm.fr/rkcl/rkcl-core

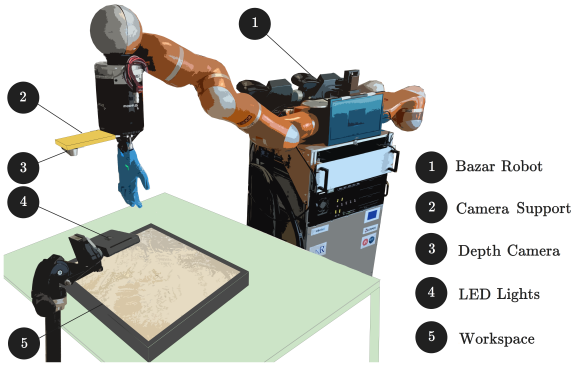


Fig. 8: Experimental Setup for robotic flour shaping.

any kind of single and multi robot structures, as long as the resulting system has a kinematic tree structure. With RKCL, the BAZAR components used in this experiment (one KUKA LWR 4+ arm and the attached Shadow hand) can be controlled in the operational or in the joint space as a unique entity. To this end, RKCL integrates a Quadratic Programming controller for tracking a desired pose with the robot tool, while respecting some constraints (e.g., joint position/velocity/acceleration limits, restrictions in the Cartesian space to avoid collisions, tool velocity/acceleration).

The grasping action is depicted in Fig. 9: the robot performs six successive operations to accomplish the whole action. A similar approach has been taken for the two other actions. To safely manage contact with the environment, we monitor the KUKA LWR's joint torques and switch to the next operation if they pass a given threshold. After each action, the arm returns to a fixed pose such that the camera is perpendicular to the workspace. There, the Realsense captures a depth image, at a distance of approximately 450 mm from the flour. This value corresponds to the depth of all actions' positions: $Z = 450$ mm. Since it is much greater than the depth of the three bounding boxes, $\Delta Z = 50$ mm (see Table I), Hypothesis 3 is valid.

We record sequences of depth images $\mathbf{L}$ of the flour, while the actions are applied by either BAZAR or a human. Images are acquired by Intel RealSense SDK 2.0, with temporal filtering and hole filling enabled[7]. We remove all images where either the robot or human are occluding the environment (typically, while the robot/human is moving to execute the action). We apply (24) to convert each image $\mathbf{L}$ into $\mathbf{I}$. We divide the recorded images into 6 datasets (one for each of the 3 actions, applied by either robot or human). All datasets are publicly available at the following link: https://seafile.lirmm.fr/d/bc9b413e83184505aa7e/. Each dataset contains 100 pairs of prior/posterior images, representing the flour before/after the action. Then, we manually annotate, in all these images, the ROI which contains a "visible" (by human standards) difference in depth[8]. For each action, we fix the same ROI sizes for the human and robot, so that the generators can be trained/tested across agents. The sizes $(\underline{w}, \underline{h})$ of the grasp, knock and poke ROI are respectively: $(183, 183)$, $(112, 60)$, $(40, 45)$; examples of these ROI are also shown in Table I.

To implement both generators $\mathbf{f}_{\mathrm{C}}$ and $\mathbf{f}_{\mathrm{DC}}$, we use Tensorflow 2 [9] on a Dell Precision 7550 laptop (Intel Xeon W-10885M CPU, 64GB RAM, NVIDIA RTX 5000). We train both networks with the Adam optimizer [36], setting $\alpha = 0.0002$, $\beta_1 = 0.5$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

B. Results of Image Prediction

First, to assess our distance d, we have compared the time needed to compute the distance between a pair of depth images, using: our distance, Chamfer distance and Earth Mover's Distance (EMD). The results (on Dell Precision 7550 mentioned above) are: our distance takes 17 ms, Chamfer 153 ms and EMD 157 ms[10]. These results confirm our choice. It should also be noted that both Chamfer and EMD rely on point clouds, with the drawbacks on memory usage mentioned above.

Then, we realized a series of experiments to assess the functions $\mathbf{f}_{\mathrm{D}}$, $\mathbf{f}_{\mathrm{CR}}$ and $\mathbf{f}_{\mathrm{DCR}}$. We split each of the 6 datasets into training+validation (75%) and testing (25%) subsets. For cross-validation, the partitions are done four times, each time selecting randomly – as test subset – a different 25% of the whole dataset. The labeled (training+validation) set $\mathcal{G}$ is used to: train generators $\mathbf{f}_{\mathrm{C}}$ and $\mathbf{f}_{\mathrm{DC}}$, find the average difference $\Delta\underline{\mathbf{I}}$, boolean matrix $\underline{\mathbf{B}}$, and mean offsets μ_{C} and μ_{D}. We train a different generator: for each action (three), agent (two: human or robot), task (two: predicting with $\mathbf{f}_{\mathrm{C}}$ or denoising with $\mathbf{f}_{\mathrm{DC}}$), and partition (four) adding up to a total 48 networks. For each network, we set the training duration to 500 epochs (approximately 1 hour).

Figure 10 shows the distances (in mm) between the predicted and true ROI (mean and standard deviation of the 4 tests) for each action and for each choice of $\mathbf{f}$ (either $\mathbf{f}_{\mathrm{D}}$, $\mathbf{f}_{\mathrm{CR}}$ or $\mathbf{f}_{\mathrm{DCR}}$). The distances are obtained by expressing distance d according to (25), and then converting it to mm. We repeat the measures four times (graphs a-d) by training and testing on either human or robot.

We can draw various conclusions from these graphs. First, testing after having trained with the same agent (Figures 10a and 10d) provides lower error than transferring from one agent to the other (Figures 10b and 10c); on average, 1.75 ± 0.42 mm versus 2.53 ± 0.46 mm. This is because the images of human and robot actions are quite different, also due to the hand sizes. Yet, the transfer learning error is low enough to encourage some of the real robot tests, which we will present in the next Section. Second, training and testing on robot (1.59 ± 0.41 mm) outperforms training and testing on human (1.92 ± 0.43 mm), likely because of the machine's lower variability. Third, independently from the agent and prediction strategy $\mathbf{f}$, the error (both mean and standard deviation) tends to increase from poke to knock to grasp. This is because of their difference in terms of repeatability (e.g., the effect of a poke, which

[7] See post-processing filters in Intel RealSense SDK 2.0: https://github.com/IntelRealSense/librealsense

[8] The annotation could be automated using standard image processing techniques, similar to the ones we developed in [5].

[9] https://www.tensorflow.org

[10] We used the following implementations: https://www.tensorflow.org/graphics/api_docs/python/tfg/nn/loss/chamfer_distance and https://en.wikipedia.org/wiki/Earth_mover's_distance#PyTorch_Implementation, for Chamfer and EMD (respectively).

This article has been accepted for publication in IEEE Transactions on Robotics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TRO.2023.3288836

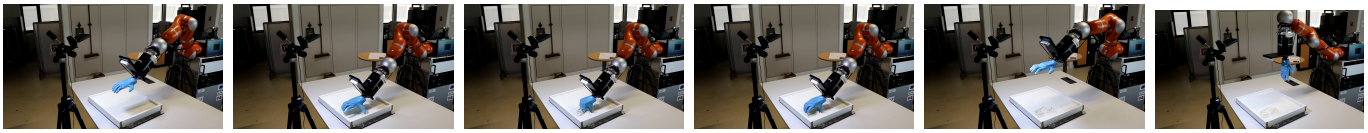
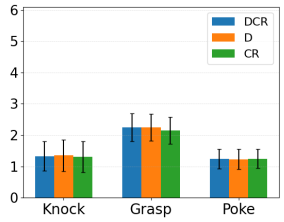
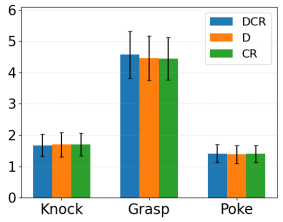


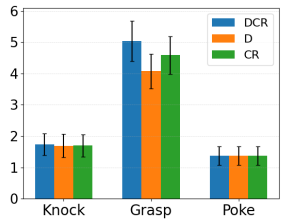
Fig. 9: Six snapshots taken during the grasp action. Left to right: the arm moves above the desired location; the arm goes down, while the hand opens; the hand closes to grasp some flour; the hand opens to release the flour; the arm goes up; the arm returns to the starting position, to free the camera field of view.



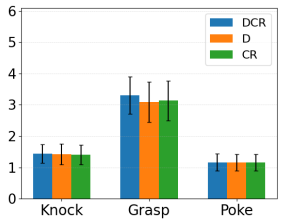
(a) Train on R and test on R.



(b) Train on R and test on H.



(c) Train on H and test on R.



(d) Train on H and test on H.

Fig. 10: Mean and standard deviation of the distances (in mm) between the predicted and true ROI, when using $\mathbf{f}_{\mathrm{D}}$ (orange), $\mathbf{f}_{\mathrm{CR}}$ (green) or $\mathbf{f}_{\mathrm{DCR}}$ (blue). The four graphs are obtained by training and testing on either human or robot data (denoted respectively H and R).

involves only one finger, varies less than that of a grasp). Last, but not least, there is no relevant difference between the mean and standard deviation of the three prediction functions, $\mathbf{f}_{\mathrm{D}}$ (orange), $\mathbf{f}_{\mathrm{CR}}$ (green) or $\mathbf{f}_{\mathrm{DCR}}$ (blue). Overall, the three functions perform similarly: $\mathbf{f}_{\mathrm{D}}$ with error 2.09 ± 0.44 mm, $\mathbf{f}_{\mathrm{CR}}$ with error 2.13 ± 0.44 mm, $\mathbf{f}_{\mathrm{DCR}}$ with 2.21 ± 0.44 mm.

To better discriminate between the three prediction functions, we have run tests with training+validation subsets of 5, 10, 25, 50 and 75 images, while keeping the same testing subset of 25 images. We consider robot data for both training and testing and average over all three actions. The results – plotted in Fig. 11 – show that as the dataset increases, the deep learning methods $\mathbf{f}_{\mathrm{DCR}}$ and $\mathbf{f}_{\mathrm{CR}}$ decrease the error faster than the more naïve $\mathbf{f}_{\mathrm{D}}$. This is likely because, with more data, the image output by $\mathbf{f}_{\mathrm{D}}$ tends to blur, while the generator embedded in $\mathbf{f}_{\mathrm{DCR}}$ and $\mathbf{f}_{\mathrm{CR}}$ succeed in capturing the depth image details.

Finally, we have assessed the importance of the refining step, by comparing the prediction error with and without refinement. More specifically, we have compared the errors of $\mathbf{f}_{\mathrm{C}}$, $\mathbf{f}_{\mathrm{CR}}$, $\mathbf{f}_{\mathrm{DC}}$ and $\mathbf{f}_{\mathrm{DCR}}$ in Table III. The results clearly show the usefulness of the refinement step (values in the third and fifth column are all lower than those in the second and fourth column).

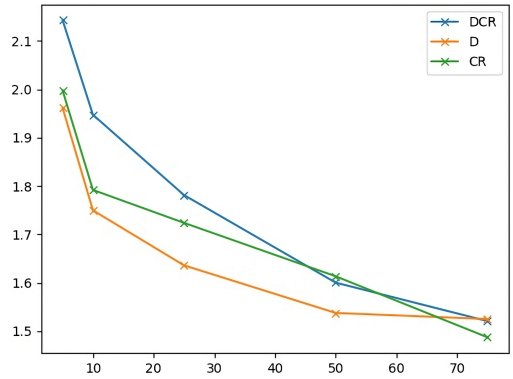


Fig. 11: Mean error (in mm) of $\mathbf{f}_{\mathrm{D}}$ (orange), $\mathbf{f}_{\mathrm{CR}}$ (green) or $\mathbf{f}_{\mathrm{DCR}}$ (blue) over the three actions as the training+validation dataset increases: 5, 10, 25, 50 and 75 images.

TABLE III: Mean error (in mm) of $\mathbf{f}_{\mathrm{C}}$, $\mathbf{f}_{\mathrm{CR}}$, $\mathbf{f}_{\mathrm{DC}}$ and $\mathbf{f}_{\mathrm{DCR}}$.

Datasets	$\mathbf{f}_{\mathrm{C}}$	$\mathbf{f}_{\mathrm{CR}}$	$\mathbf{f}_{\mathrm{DC}}$	$\mathbf{f}_{\mathrm{DCR}}$
Train on R and test on R	1.65	1.56	1.73	1.60
Train on R and test on H	2.68	2.51	2.80	2.55
Train on H and test on R	2.70	2.55	2.95	2.71
Train on H and test on H	1.96	1.90	2.11	1.97

C. Robotic point cloud shaping

Since among the three prediction functions, $\mathbf{f}_{\mathrm{CR}}$ appears to be the most promising as data increases (see Fig. 11), we have used it in the following experiments to predict the effect of an action on a region of interest.

We have run two series of experiments: the first four with robot-generated images and the next two with human-generated images. In both series, for each of the three actions, we take the best network $\mathbf{f}_{\mathrm{C}}$ among the four dataset partitions, and then proceed as follows. Given a pair of images ($\mathbf{L}_0$ of the initial point cloud $\mathcal{P}_0$, and $\mathbf{L}_*$ of the final desired point cloud $\mathcal{P}_*$), we run **Algorithm 1** (with $\mathbf{f}_{\mathrm{CR}}$ applied at line 7) to obtain the sequence of actions $\mathbf{s}_*$ for shaping the point cloud from $\mathcal{P}_0$, to $\mathcal{P}_*$. Then, we control BAZAR robot, so that it sequentially executes the actions in $\mathbf{s}_*$, starting from point cloud $\mathcal{P}_0$. The final desired depth images $\mathbf{L}_*$ have been shaped by the robot in the first series of experiments, and by the human in the second series.

The results are shown in Fig. 12. For each of the six experiments, we show (from left to right) the initial image $\mathbf{L}_0$, the final desired image $\mathbf{L}_*$, the image predicted by **Algorithm 1**, and the image obtained by the robot; we also indicate the computed sequence $\mathbf{s}_*$ and the evolution of d (in mm) over the experiments. In all six experiments, **Algorithm 1** retrieves the actions which were applied by the agent to obtain $\mathbf{L}_*$, although not necessarily in the same order. As the robot replays the sequence, the distance diminishes at each step in all

experiments (see blue curves). As expected, the final distance is higher in the transfer learning experiments (both dataset and desired image are human-made) than in the entirely robotic experiments. This is due to the difference in shape and size of the human and robot actions, which is visible by comparing the fourth image to the second and third images, in the last two experiments.

D. Generalizing across actions, materials, and geometry

Given the limitations of the robot hardware, we decided to run another series of experiments on human data, to validate our methodology in more general situations. In particular, we assessed the following aspects:

- applying any among all five actions of Table I (including pinch and press),
- shaping new – non granular – material with the functions $\mathbf{f}_{\mathrm{D}}$, $\mathbf{f}_{\mathrm{CR}}$ trained on flour; to this end, we used kinetic sand, a plastic toy material that mimics the physical properties of wet sand,
- experimenting action superposition (i.e., situations where multiple actions affected the same region),
- shaping non-flat surfaces (both irregular and of constant, non-null curvature).

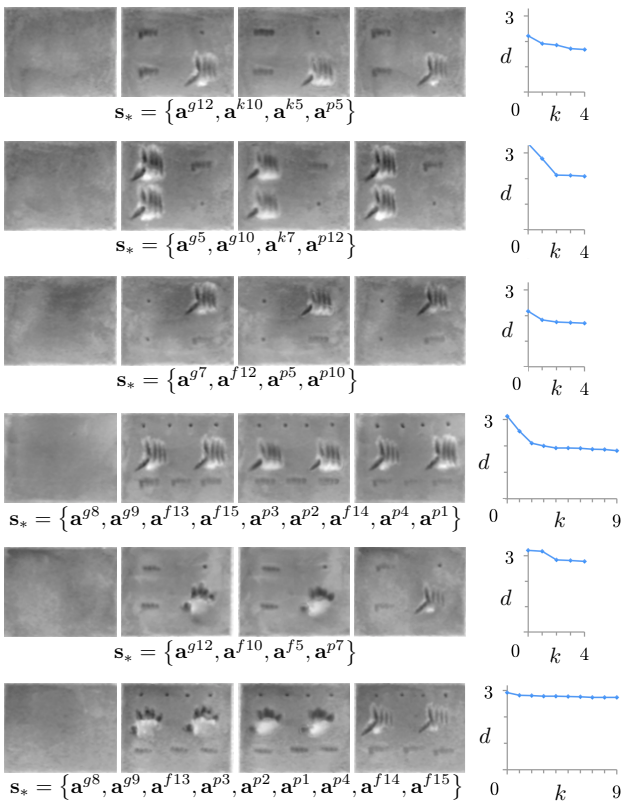


Fig. 12: Six robot experiments, with the neural networks trained on the robot dataset (top four) and on the human dataset (bottom two). For each of the six experiments, we show four images (left to right: initial image $\mathbf{L}_0$, final desired image $\mathbf{L}_*$, predicted image, and image obtained by the robot) as well as the computed sequence $\mathbf{s}_*$ and the evolution of d (in mm) after each action $\mathbf{a}_k$ is applied.

We have run five experiments on kinetic sand, shown in Fig. 13, with $\mathbf{f}_{\mathrm{CR}}$ as prediction function. The actions can be applied in $N = 21$ positions; hence, for these experiments, the total number of actions is $\dim \mathcal{A} = 5N = 105$. In all experiments and for each of the five actions, we take the best network $\mathbf{f}_{\mathrm{C}}$ among the four human dataset partitions, and then proceed as follows. Given a pair of images ($\mathbf{L}_0$ of the initial point cloud $\mathcal{P}_0$, and $\mathbf{L}_*$ of the final desired point cloud $\mathcal{P}_*$), we run **Algorithm 1** (with $\mathbf{f}_{\mathrm{CR}}$ applied at line 7) to obtain the sequence of actions $\mathbf{s}_*$ for shaping the point cloud from $\mathcal{P}_0$, to $\mathcal{P}_*$. Then, we verify if the predicted image qualitatively resembles the final one, and whether **Algorithm 1** predicts a sequence of actions similar to the actually applied one.

For each of the 5 experiments, we show (from left to right) the initial image $\mathbf{L}_0$, the final desired image $\mathbf{L}_*$ and the image predicted by **Algorithm 1**; we also indicate the computed sequence $\mathbf{s}_*$ and the evolution of d (in mm) over the experiments.

In the first experiment, the initial shape has a constant curvature of approximately 15 cm. For the four other experiments, we start with very irregular non-flat shapes (with standard variation of the depth from its mean ± 17 mm – see first column of Fig. 13). In the fourth and fifth experiments, we obtained the desired image by apply overlapping actions.

In four of the five experiments, **Algorithm 1** retrieves the same type, position and number of actions which the human applied to obtain $\mathbf{L}_*$, although not necessarily in the same

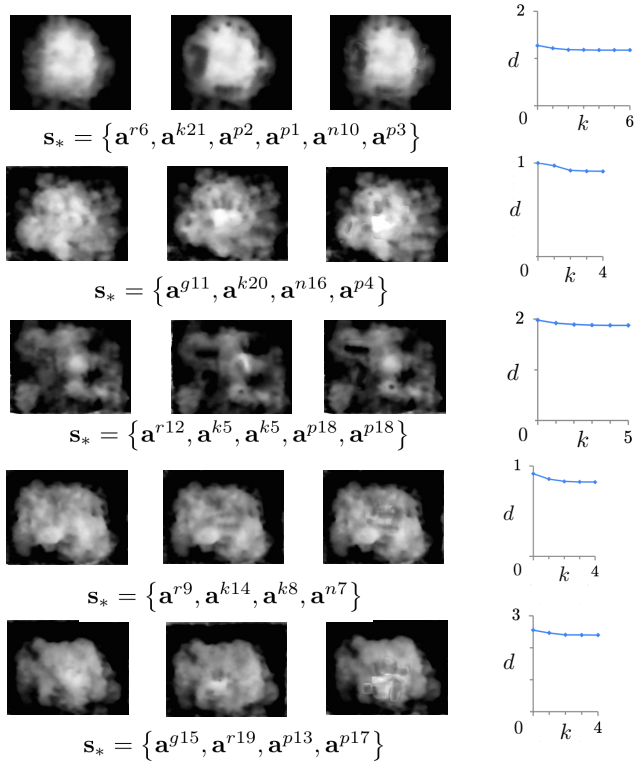


Fig. 13: Five experiments, with the neural networks trained on the human dataset. For each experiment, we show three images (left to right: initial image $\mathbf{L}_0$, final desired image $\mathbf{L}_*$ and predicted image) as well as the computed sequence $\mathbf{s}_*$ and the evolution of d (in mm) after each action $\mathbf{a}_k$ is applied.

order. The only exception is the third experiment. Here, our method predicts two knock and two poke actions at the same position (see $\mathbf{s}_*$ at the third line of Fig. 13), whereas the applied sequence was: $\mathbf{s}_* = \{\mathbf{a}^{r12}, \mathbf{a}^{k5}, \mathbf{a}^{p18}\}$. When applying knock and poke to generate $\mathbf{L}_*$, the human has penetrated the material more than usual. On the human dataset, the deepest points generated by knock and poke have depths respectively 6.80 mm and 7.05 mm. Instead, to generate $\mathbf{L}_*$, the human applied a knock and a poke of maximal depths 19.60 mm and 17.05 mm, respectively. Therefore, the solution proposed by **Algorithm 1** is coherent with the task. It should also be noted that in all five experiments, the distance diminishes at each step (see blue curves).

Finally, we repeated the five experiments using the naïve $\mathbf{f}_\mathrm{D}$ function to predict action effect. The results were much less satisfying in terms of both final image quality and distance error. In none of the experiments, **Algorithm 1** managed to retrieve the type, position and number of actions which the human applied to obtain $\mathbf{L}_*$. This result leads us to believe that the molding problem cannot be solved by such a simple approach, and that particularly in complex scenarios such as the ones studied here, deep learning can provide a much better solution to the action prediction problem.

VII. CONCLUSIONS AND FUTURE WORK

Although our results are encouraging, they are preliminary – to say the least. The path towards robotic molding is still long and paved with obstacles.

First, the current hardware substantially limits the range of robot operation. Even an extraordinary robot like the Shadow Dexterous Hand, used here, is designed for conventional rigid object manipulation, and can by no means reproduce the versatility and adaptability of the human hand. It is rigid, has non-reproducible behavior (due to cable actuation), is weak (e.g., not strong enough to penetrate compressed flour) and tends to break easily. This is also why if was difficult to actuate the fingers, and in two out of three robot actions (all except *grasp*) we maintained a constant hand posture while in contact with the material. We had to dedicate most of our time to find the appropriate material and actions, in order to record relevant data without breaking the robot! Also in terms of perception, the gap between technology and nature is critical.

Present-day force sensing is far from providing good accuracy at the low ranges required by our application, and it could not be used to close the actions' feedback loop (e.g., to decide when or how to grasp the flour). Our Shadow Hand is equipped with five (one per fingertip) SynTouch LLC Biotac tactile sensors. Yet, these sensors are uncalibrated and only provide information if the fingertip touches the environment, which is rarely the case when molding. Furthermore, to shield the robot mechanics and the Biotac themselves from flour, the hand wears a glove, which biases the Biotac readings. The ideal sensor for molding should cover palm and fingers, and be dirt and waterproof. For all these reasons, although we acknowledge the importance of force sensing, we have only focused on visual feedback.

In the future, solutions to these actuation and perception issues may come from soft [37] and underactuated [38] robots or from dense tactile skins [39]. Another worthy aspect is that humans generally rely on both hands for molding; yet, this introduces other difficulties, beyond the scope of this paper

A second limitation comes from the absence of a physical model, which led us to use deep machine learning. We could improve our deep learning methods by adding data from multiple humans (not just one, like here), relying on better computational resources, or automating image annotation as in [5]. Yet, our results show that deep learning outperforms image difference in generalizing across actions, materials and geometry (e.g., irregular surfaces). One of the limitations of the neural networks developed here is that they are not scalable in terms of image size, and therefore lack robustness with respect to variations of the action depth coordinate. Due to their synthetic nature, the predicted images may contain some artifacts and discontinuities. Yet, despite these artifacts, our framework could compute the correct sequence of actions.

A third issue, outlined above, is that our framework relies on too many assumptions, which should gradually be relaxed, to make it applicable in real scenarios. For instance, we cannot afford variations in the relative pose between the camera and workspace, unless a rigorous calibration phase is carried out systematically, to guarantee consistency between the robot operational space and the camera frame. Besides, our action space is limited to predetermined actions. This raises the question of whether our method could provide a reasonable prediction for unknown actions. Furthermore, the action sequence search problem requires the actions to be applied at a limited and known set of positions, making it easily intractable in ambitious scenarios. To generalize to continuous action positions, one could use random sampling, Monte Carlo tree search or particle swarm optimization, to select the best sequence of actions/positions. These approaches are more general than greedy ones, since they explore a larger space of possible actions sequences. Yet, they can be computationally expensive, especially if the search space is large and the prediction model complex (as is the case here).

Despite the above issues, there is still room for hope. For instance, we believe that our approach – transforming the prediction problem and the distance metric from 3D point cloud to depth image space – is a promising avenue of research for soft plastic object manipulation. First, it can be enriched by 2D image processing techniques (e.g., convolution, mutual information). Second, in contrast with state-of-art methods (such as Iterative Closest Point), it guarantees the fast computation and low memory usage required for robot molding – at both learning and planning steps. Third, it could encompass a coarse mechanical model of the object. Generally, in the authors' view, the future of soft plastic object manipulation lies in the successful integration of a coarse physical model, refined with data-based techniques.

ACKNOWLEDGMENTS

The authors would like to thank Benjamin Navarro for his help on the robot programming and André Crosnier, Fabrice Caussidery, Phlippe Fraisse, Hervé Louche, Serge Mora, Jean-Michel Muracciole for their suggestions on the material to be used in the experiments.

REFERENCES

[1] S. Duenser, R. Poranne, B. Thomaszewski, and S. Coros, "Robocut: Hot-wire cutting with robot-controlled flexible rods," *ACM Trans. Graph.*, vol. 39, no. 4, aug 2020.

[2] Z. Ma, S. Duenser, C. Schumacher, R. Rust, M. Bacher, F. Gramazio, M. Kohler, and S. Coros, "Stylized robotic clay sculpting," *Computers Graphics*, vol. 98, pp. 150–164, 2021.

[3] N. Xuejuan, L. Jingtai, S. Lei, L. Zheng, and C. Xinwei, "Robot 3D sculpturing based on extracted nurbs," in *IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, 2007.

[4] L. Pagliarini and H. Hautop Lund, "The development of robot art," *Artificial Life and Robotics*, vol. 13, no. 2, pp. 401–405, 2009.

[5] A. Cherubini, V. Ortenzi, A. Cosgun, R. Lee, and P. Corke, "Model-free vision-based shaping of deformable plastic materials," *Int. Journal of Robotics Research*, vol. 39, no. 14, pp. 1739–1759, 2020.

[6] F. Chaumette and S. Hutchinson, "Visual servo control, Part I: Basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[7] A. Anand, H. S. Koppula, T. Joachims, and A. Saxena, "Contextually guided semantic labeling and search for three-dimensional point clouds," *Int. Journal of Robotics Research*, vol. 32, no. 1, pp. 19–34, 2013.

[8] H. Hwang, S. Hyung, S. Yoon, and K. Roh, "Robust descriptors for 3d point clouds using geometric and photometric local feature," in *IEEE/RSJ Int. Conf. on Robots and Intelligent Systems*, 2012.

[9] F. Engelmann, T. Kontogianni, J. Schult, and B. Leibe, "Know what your neighbors do: 3d semantic segmentation of point clouds," in *Computer Vision –ECCV 2018 Workshops*, L. Leal-Taixé and S. Roth, Eds. Springer International Publishing, 2019, pp. 395–409.

[10] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Yale-cmu-berkeley dataset for robotic manipulation research," *Int. Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.

[11] J. Sung, S. H. Jin, and A. Saxena, "Robobarista: Object part based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds," in *Int. Symposium on Robotics Research*, 2015.

[12] A. Pas and R. Platt, *Using Geometry to Detect Grasp Poses in 3D Point Clouds*. Springer International Publishing, 01 2018, pp. 307–324.

[13] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg, "Dex-net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning," in *IEEE Int. Conf. on Robotics and Automation*, 2018.

[14] I. Ziamtsov and S. Navlakha, "Machine learning approaches to improve three basic plant phenotyping tasks using three-dimensional point clouds," *Plant Physiology*, vol. 181, no. 4, pp. 1425–1440, 2019.

[15] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," *Int. Journal of Computer Vision*, vol. 13(12), pp. 119–152, 1994.

[16] N. Fioraio, K. Konolige, and W. Garage, "Realtime visual and point cloud slam nicola fioraio willow garage," in *Proc. of the RGB-D workshop on advanced reasoning with depth cameras at Robotics: Science and Systems (RSS)*, 2011.

[17] Y. Liu, C. Wang, Z. Song, and M. Wang, "Efficient global point cloud registration by matching rotation invariant features through translation search," in *European Conference on Computer Vision*, 2018.

[18] H. Yang, J. Shi, and L. Carlone, "TEASER: Fast and certifiable point cloud registration," *IEEE Trans. on Robotics*, vol. 37, no. 2, pp. 314–333, 2021.

[19] J. Ma, J. Wu, J. Zhao, J. Jiang, H. Zhou, and Q. Z. Sheng, "Non-rigid point set registration with robust transformation learning under manifold regularization," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 30, no. 12, pp. 3584–3597, 2019.

[20] R. A. Newcombe, D. Fox, and S. M. Seitz, "DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time," in *2015 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 343–352.

[21] F. F. Khalil, P. Curtis, and P. Payeur, "Visual monitoring of surface deformations on objects manipulated with a robotic hand," in *IEEE Int. Workshop on Robotic and Sensors Environments (ROSE)*, 2010.

[22] M. Staffa, S. Rossi, M. Giordano, M. De Gregorio, and B. Siciliano, "Segmentation performance in tracking deformable objects via WNNs," in *IEEE Int. Conf. on Robotics and Automation*, 2015, pp. 2462–2467.

[23] J. Schulman, A. Lee, J. Ho, and P. Abbeel, "Tracking deformable objects with point clouds," in *IEEE Int. Conf. on Robotics and Automation*, 2013.

[24] S. Elliott and M. Cakmak, "Robotic cleaning through dirt rearrangement planning with learned transition models," in *IEEE Int. Conf. on Robotics and Automation*, 2018.

[25] C. Schenck, J. Tompson, D. Fox, and S. Levine, "Learning robotic manipulation of granular media," in *Conf. on Robot Learning (CoRL)*, 2017.

[26] Y. Li, J. Wu, R. Tedrake, J. Tenenbaum, and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids," *arXiv preprint arXiv:1810.01566*, 2018.

[27] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3d object reconstruction from a single image," *CoRR*, vol. abs/1612.00603, 2016. [Online]. Available: http://arxiv.org/abs/1612.00603

[28] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[29] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CVPR*, 2017.

[30] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 9(1), pp. 62–66, 1979.

[31] V. Jain and H. S. Seung, "Natural image denoising with convolutional networks," in *Advances in Neural Information Processing Systems*, vol. 21, 2009, pp. 769–776.

[32] M. Moll and L. E. Kavraki, "Path planning for deformable linear objects," *IEEE Trans. on Robotics*, vol. 22, no. 4, pp. 625–636, 2006.

[33] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.

[34] M. Laskey, C. Powers, R. Joshi, A. Poursohi, and K. Goldberg, "Learning robust bed making using deep imitation learning with DART," *arXiv preprint arXiv:1711.02525*, 2017.

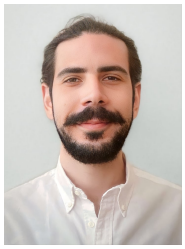
[35] A. Cherubini, R. Passama, B. Navarro, M. Sorour, A. Khelloufi, O. Mazhar, S. Tarbouriech, J. Zhu, O. Tempier, A. Crosnier, P. Fraisse, and S. Ramdani, "A collaborative robot for the factory of the future: Bazar," *Int. Journal of Advanced Manufacturing Technology*, vol. 105(9), pp. 3643—3659, December 2019.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd Int. Conf. on Learning Representations, ICLR*, 2015.

[37] J. Shintake, V. Cacucciolo, D. Floreano, and H. Shea, "Soft robotic grippers," *Advanced Materials*, vol. 30(29), p. 1707035, 2018.

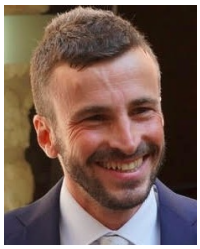
[38] C. Della Santina, G. Grioli, M. Catalano, A. Brando, and A. Bicchi, "Dexterity augmentation on a synergistic hand: The pisa/iit softhand+," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 497–503.

[39] Z. Kappassov, J.-A. Corrales, and V. Perdereau, "Tactile sensing in dexterous robot hands -— review," *Robotics and Autonomous Systems*, vol. 74, pp. 195–220, 2015.



Ege Gursoy received the M.Sc. in Robotics in 2020 from Université de Montpellier, France. From 2020 to 2021 he worked at UM as a research engineer in the IDH (Interactive Digital Humans) research group. Currently, he is a PhD candidate in a collaboration between Université de Montpellier and Monash University, Australia.

Sonny Tarbouriech received the M.Sc. in Electrical and Computer Science Engineering in 2014 from the École Polytechnique de Montpellier and a second M.Sc. in Robotics in 2016 from the University of Sherbrooke, Canada in 2016. From 2016 to 2019, he worked at Tecnalia and was a PhD student at Université de Montpellier, under the supervision of Prof. Philippe Fraisse. Since graduation, Sonny has been working at UM as a research engineer in the IDH group.



Andrea Cherubini received the M.Sc. in Mechanical Engineering in 2001 from the University of Rome La Sapienza and a second M.Sc. in Control Systems in 2003 from the University of Sheffield, U.K. In 2008, he obtained the Ph.D. in Control Systems from La Sapienza, and started a three year postdoc at INRIA Rennes. Since 2011, he is at Université de Montpellier, first as Associate Professor, and then as Full Professor. At UM, he is in charge of the Robotics Master, and leads the IDH (Interactive Digital Humans) research group.