VISION BASED SHAPING OF DEFORMABLE MATERIALS

Master 2 EEA parcours Robotique

Rapport de Stage

Année scolaire: 2019/2020

Etudiant: Ege GURSOY

Encadrant: Andrea CHERUBINI

Table of Contents

I. Introduction

In this project, we aim at shaping deformable materials with a combination of robot arm and hand, using the visual feedback.

Given a desired shape of the object and a set of predefined actions, we aim at predicting the effects of actions using an artificial neural network architecture, creating a tree from all possible combinations and finding the shortest sequence of actions to drive the initial shape of the object to the desired one, with the visual feedback.

Since the visual information does not rely on the dynamic properties of the material, it should work without prior knowledge about the dynamic model of the object.

II. Related Work

The literature on the manipulation of deformable objects is not as common as that of rigid objects. It is reviewed under two main topics: deformation control and action selection. Most of this section is extracted from the work of Cherubini et al. (2019) which studied a very similar subject.

1.Deformation control

The works that control the object's deformation can be classified in two categories: those that require a model and the data-driven ones (model-free).

Model based approaches take advantage of the deformable object's physical model, whenever it is available. Howard and Bekey (2000) train a neural network on a physics-based model to extract the minimum force required for 3D object manipulation. Gopalakrishnan and Goldberg (2005) focus on 2D objects and use a mesh model with linear elastic polygons. Their approach, called "deform closure" is an adaptation of "form closure", a well-known method for rigid object grasping. Das and Sarkar (2011) use a mass-spring-damper model to simulate a planar object so that its shape, described by a curve, can be changed into another desired shape. Their method, however, requires a significant number of actuation points (over 100), and is only validated in simulations. Higashimori et al. (2010) use a four-element model and present a two-step approach where the elastic parameters are estimated by force sensing, and then the force required to reach the desired shape is calculated based on

the plastic response. Cretu et al. (2012) monitor shape deformation by tracking lines that form a grid on the object. A feed-forward neural network is used to segment and monitor the deformation. Arriola-Rios and Wyatt (2017) predict the object behavior by first classifying the material, then using force and computer vision to estimate its plastic and elastic deformations. In Ficuciello et al. (2018), a method for dexterous in-hand manipulation of 3D soft objects for real-time deformation control is presented, relying on Finite Element Modeling. However, the authors assume the object to be purely elastic.

Model-free approaches explored deformable object manipulation without explicitly modelling the deformations. In their pioneering work, Wada et al. (2001) designed a PID controller that can manipulate 2D objects in the absence of a prior model. Smolen and Patriciu (2009) use a mesh-less model of the object where a set of points are controlled on the surface. A Jacobian transform is derived and used to control the robot motion. Berenson (2013) uses the concept of diminishing rigidity to compute an approximate Jacobian of the deformable object. In his work, human and robot simultaneously manipulate the object (a 2D cloth). More recent model-free approaches rely on machine learning. Gemici and Saxena (2014) learn haptic properties such as plasticity and tensile strength of food objects. Other researchers propose to predict the next state given the current state and a proposed action. For instance, Elliott and Cakmak (2018) use some defined primitive tool action for rearranging dirt and Schenck et al. (2017) present a Convolutional Neural Network for scooping and dumping granular materials. Li et al. (2018) propose to learn a particle-based simulator for complex control tasks. This enables the simulator to quickly adapt to new environments or to unknown dynamics within a few observations. Cherubini et al. (2019) proposed optimization-based algorithm for controlling the robot along with a neural network architecture to the action to be applied using real human data. However, only two primitive actions are defined using basic tools to shape the material.

2. Action selection

When manipulation requires various types of actions, an automatic action selection method is needed. Such methods are often based on machine learning or on motion planning. Learning-based methods are frequently used for choosing within a finite number of actions. An example is deep reinforcement learning (see the works of Dulac-Arnold et al. (2015) and of Isele et al. (2018)), which relies on the Markov Decision Process (MDP) formulation (refer to Puterman (2014)). Laskey et al. (2017) learn a policy for choosing actions to manipulate cloth using imitation learning. Moll and Kavraki (2006) present a planning method for finding intermediate states and transitioning between these states to shape a flexible wire. Similarly, Zhu et al. (2018) plan the sequence of actions for shaping a cable using environmental contacts.

III. Contribution

In our method, the deformation of the material is not modelled. Thus, it is a model-free approach. Instead, the deformation information is obtained by visual changes on the material. The effects of a single deformation action are well-defined and can be outputted by training an autoencoder model, a type of ANN, with pre-recorded examples. For the rest, the algorithmic choices rely on the sequential nature of the task. A sequence of predefined actions is required to achieve a given desired shape. All the possible actions are tried to find the shortest sequence of actions which is closer to the desired shape, in terms of depth-distance.

Real, human generated data are used in the deformation control section experiment to make it applicable in the real world. In the same section, autoencoder based model is trained using these images to improve robustness.

In summary, the contributions of this work are:

1. Deformation control method relies on the real-world images. The human data is utilized to design a strategy for robotic kinetic sand manipulation and also to train a machine learning-based action model.
2. Proposed finite number of action classes to reduce the complexity of the deformable object manipulation problem.
3. Relatively complicated actions are produced using robot hand with many degrees-of-freedom.
4. Mapping between current state and action to next state, instead of current and desired state to action.
5. A scalar metric is defined for measuring the distance between two point clouds which is used as a loss function in the autoencoder to predict the effect of actions for a given input shape, a measure for finding the shortest path in the decision tree which leads to the desired shape.
6. Method is highly applicable in the real world, unlike some previously presented methods, which quickly reach their limits in real world experiments.

IV. Fundamentals of artificial neural networks

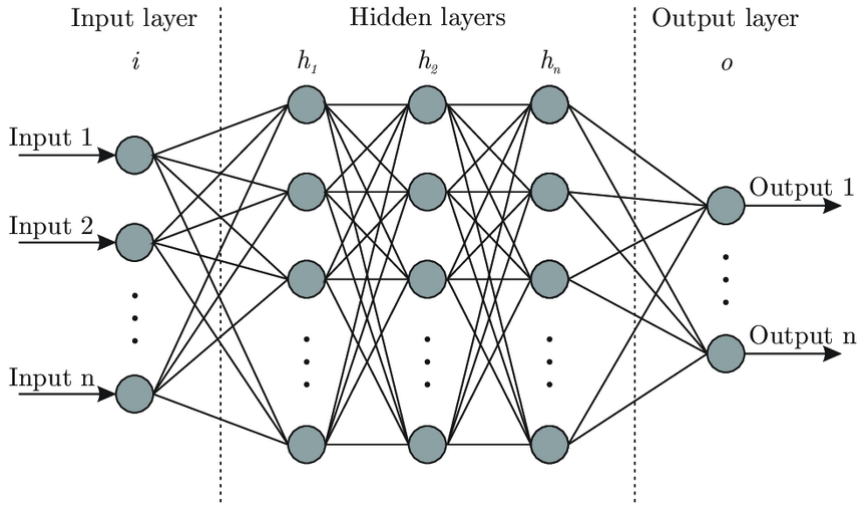

Figure 1: ANN representation

Artificial neural network (ANN) or neural network for short is an algorithm inspired by the neurons in our brain. It is designed to recognize patterns in complex data, and often performs the best when recognizing patterns in images, videos, sounds etc. Fig. 1 shows the general representation of an ANN.

A neural network consists of layers. Each layer has a number of neurons. Connections between the neurons of different layers are called weights. All the weights come with a bias number. As the number of layers increases, the network becomes deeper.

The equation of neurons is wrapped by an activation function to obtain a non-linear relation between layers.

To move forward through the network, called a forward propagation, each neuron in the next layer is calculated iteratively until an output is achieved.

A cost (loss) function C measures the similarity between the generated output and the desired output. Given the result of C, the error on each weight and bias, called gradient, is computed. This step is called a backward propagation.

An optimizer function updates the parameters in order to minimize the error of C.

This cycle is called the training phase and it continues until we get satisfactory results.

a) Optimizer

Optimizers determine the method by which the loss function is minimized.

Adaptive Moment Estimation (Adam) is an optimization method presented by Diederik Kingma and Jimmy Ba (2015) and since then it is widely used in ANNs.

Adam algorithm from the original paper is below (see fig.2):

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 2: Adam optimization algorithm

b) Activation function

The activation function of a neuron defines the output of that neuron given an input. Weights and biases pass through this function before the output. They are usually non-linear functions and are the main reason of the nonlinearity in ANNs. Otherwise, the whole network can be reduced to a linear relation.

- Rectified linear unit

Rectified Linear Unit (ReLu) is a non-linear activation function as described in fig.3:

$$f(x) = ma\,x(0, x) \qquad eq.1$$

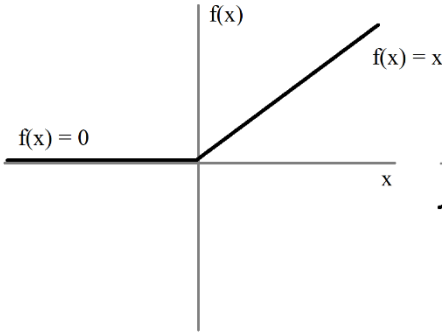

Figure 3: ReLu function

It has the advantage of having less time and space complexity. It does not evolve the exponential operation, unlike most of the functions, which are more costly. It also avoids the vanishing gradient problem which is a common problem in ANNs during the training phase. Since the backward propagation uses the derivatives to compute the gradients, if they become too small, parameters

It introduces the dead relu problem, where components of the network are most likely never updated to a new value.

- Leaky rectified linear unit

It is an activation function, a slightly modified version of ReLu, as described in fig 4:

$$f(x) = \begin{cases} x & if\ x > 0 \\ ax & otherwise \end{cases} \qquad eq.2$$

With a is a small negative number.

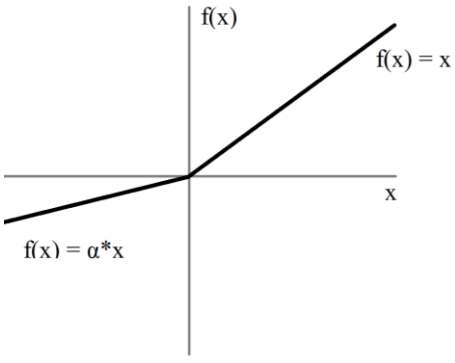

Figure 4: Leaky ReLu function

It avoids the dead relu problem, since we allow a small gradient, when computing the derivative.

c) Layer

ANNs are essentially a stack of layers each containing n number of neurons. The input layer is the first layer of the ANN which contains the raw data. The output layer is the final layer which gives the result. The layers in between are called the hidden layers. As the number of these layers get higher, the network gets "deeper".

There are different types of layers for different applications. Below are the types that we used in our network.

- Fully Connected

Fully connected (FC) layer connects every neuron in one layer to every neuron in another layer. It is the simplest and most primitive form of layer. Fig 5 shows 4 FC layers connected in series.

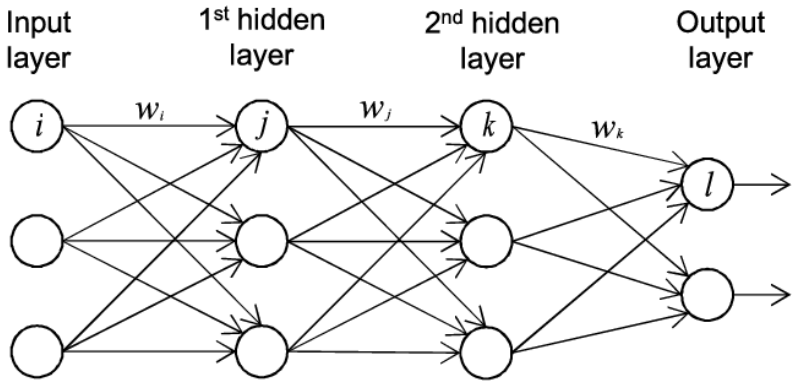

Figure 5: Example of 4 fully connected layers

- Convolution

Convolution is a linear operation that involves the multiplication of a set of weights with the input (see Dumoulin and Visin (2016)). The multiplication is performed between an input data and a matrix of weights, called a kernel. A kernel of value with a predefined size, slides with a stride across the input image. The stride defines the step size of the kernel when traversing the image. At each location, the product between each element of the kernel and the input element it overlaps is computed and the results are summed up to obtain the output in the current location. The final output of this procedure is passed to the next layer (see fig. 6)

This procedure can be repeated using different kernels to form as many outputs as desired. These outputs are called output channels.

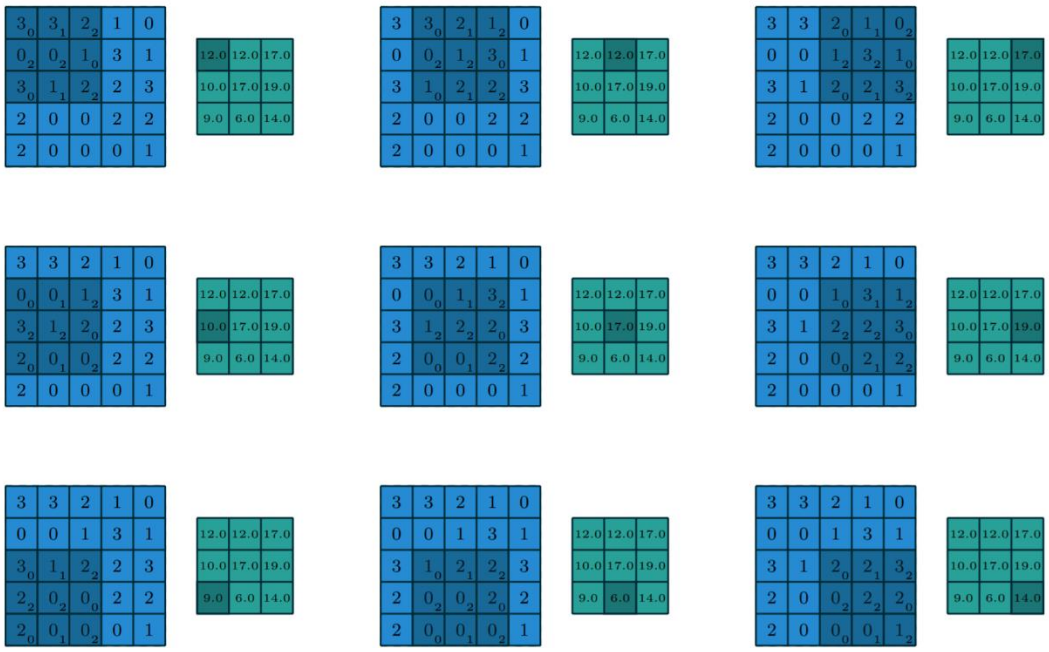

Figure 6: Convolution example with 3x3 kernel on 5x5 input with 1x1 stride

- Maxpooling

Maxpooling is a downsampling strategy to reduce the dimensions of the data by combining outputs of neuron clusters at one layer into a single neuron in the next layer (see Dumoulin and Visin (2016)). Pooling combines small clusters defined by the number of pool sizes. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer (see fig. 7).

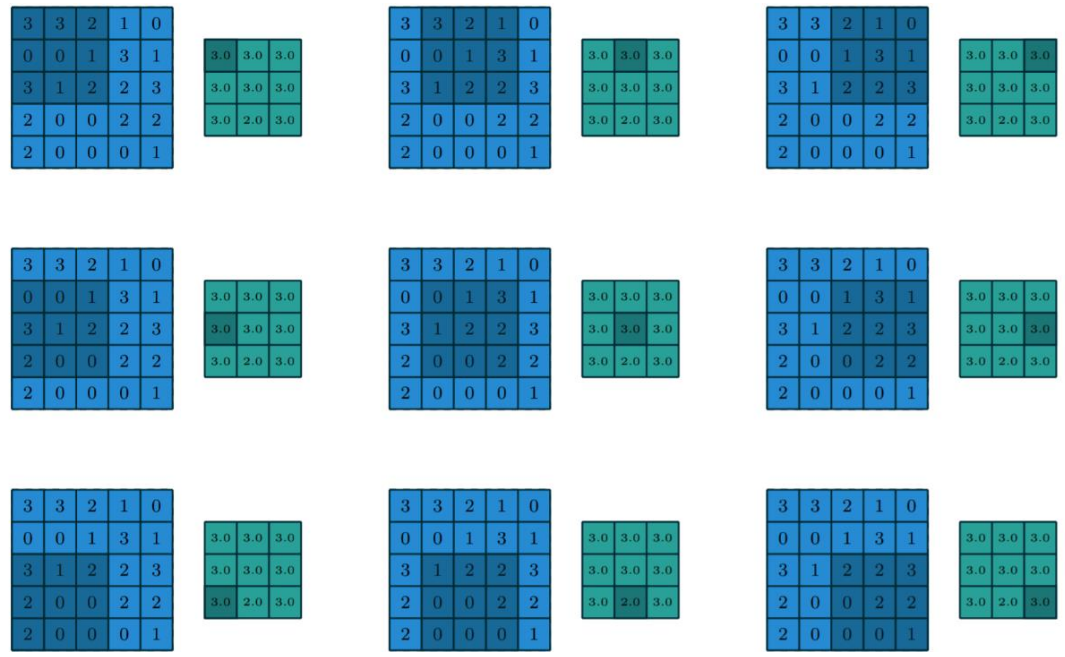

Figure 7: Maxpool example with 3x3 kernel on 5x5 input using 1x1 stride

- Padding

Padding refers to the data added to a data matrix. It defines how the borders of a sample are handled. Without a padding, we tend to lose data on the outer borders.

A half padding will keep the spatial output dimensions equal to the input by adding zero valued data equally to the border of the input data matrix (see fig. 8 and Dumoulin and Visin (2016)).

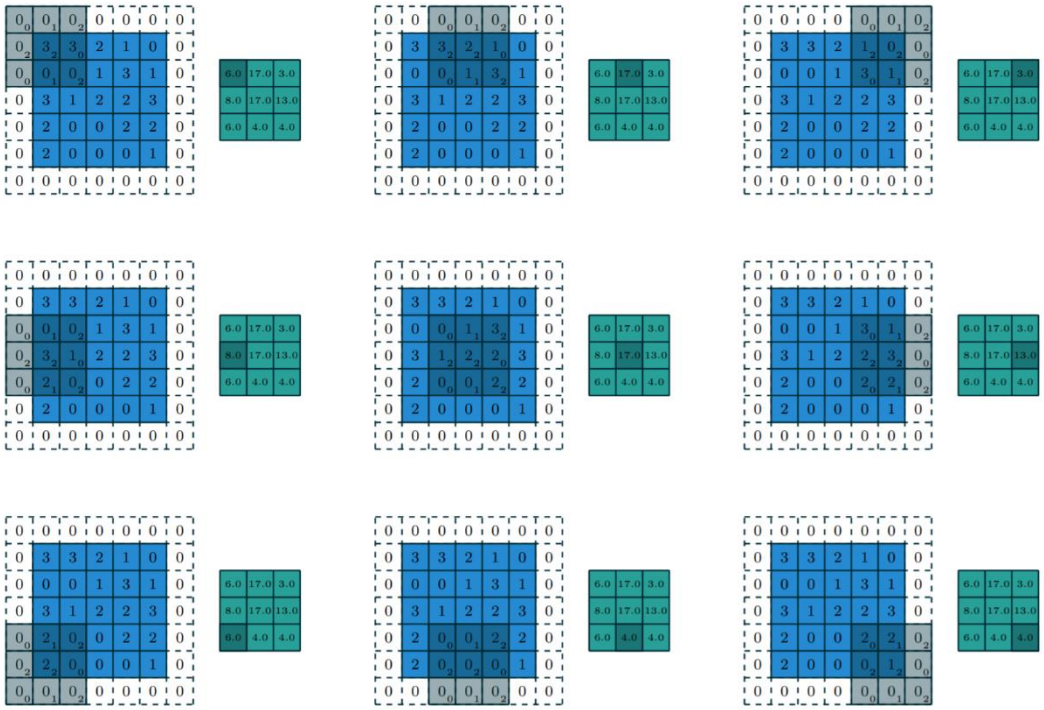

Figure 8: Convolution example with 3x3 kernel on 5x5 input using 1x1 stride and half-padding.

- Transposed Convolution

The transposed convolution carries out a regular convolution but reverts its spatial transformation. It can be used to revert the effects of the convolution.

The output dimensions are computed as the opposite of a convolution operation while still performing a normal convolution operation. To achieve this, the input is expanded with a padding before the convolution operation (see fig. 9).

To revert an image after a convolution operation with half padding, since there is no information about the amount of padding, the transposed convolution reverts the image to the worst case (i.e. with maximum padding). This may cause the output image to be bigger than the original image.

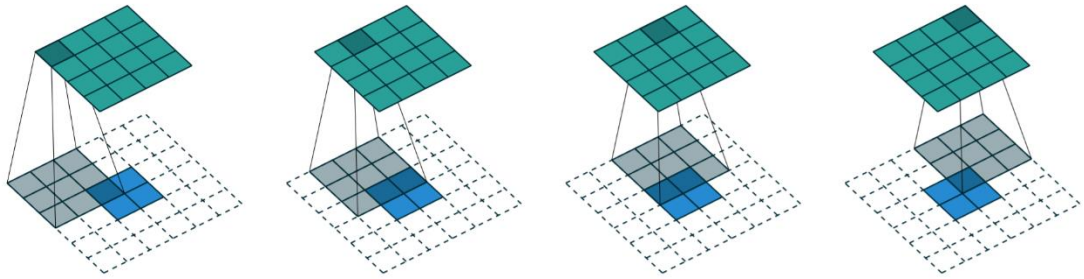

Figure 9: Transposed convolution example with 3x3 kernel on 4x4 input using 1x1 stride. Note that it is equivalent to convolving a 3x3 kernel on 2x2 input padded with 2x2 border of zeros using 1x1 strides

- Upsampling

Upsampling is one of the ways to upsample the compressed data. It is used to reverse the effect of the pooling by extending all the single data by a matrix of the same single data with predefined kernel dimensions (see fig. 10 and CS231n course by Standord).

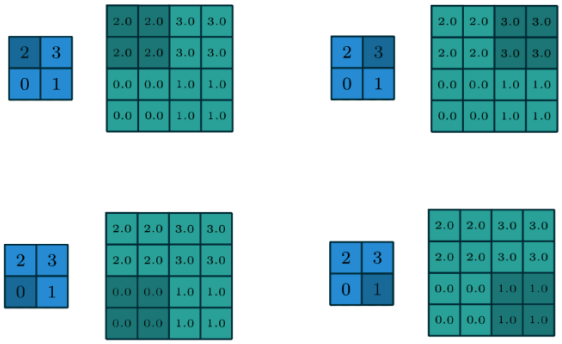

Figure 10: Upsampling example with 2x2 kernel on 2x2 input using 1x1 stride

a) Autoencoder

An autoencoder model is a type of ANN used to learn efficient data codings in an unsupervised (i.e. with unlabelled data) manner (see Kramer (1991)). The aim of an autoencoder is to learn a dimensionally reduced latent representation of a set of data. Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input. Autoencoders can be used as a generative model, more precisely, generate a new image from another given image.

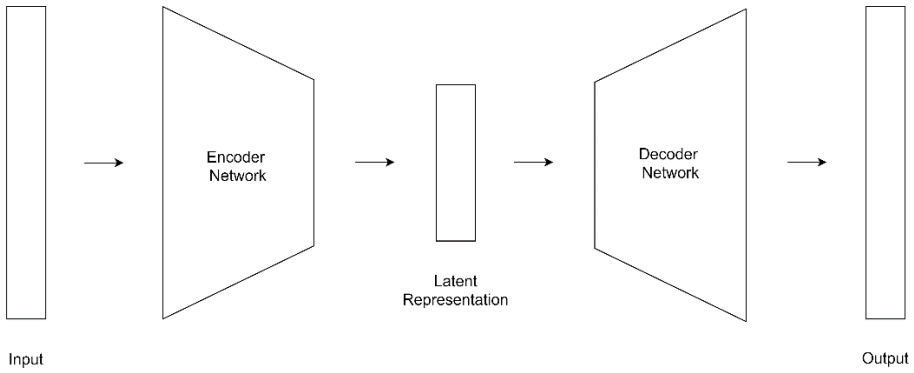

Figure 11: Autoencoder network

The autoencoder model is a combination of 2 networks. An encoder network which takes an image as input and outputs its latent space representation, is followed by a decoder network which has an input and an output in the reverse order (see fig. 11). The complete autoencoder takes an image as input, generates another one at the end.

V. Proposed method

1.Primitive actions

Humans often use non-prehensile actions to move objects or modify the environment. Such actions become even more common when the object of interest is deformable. Considering these observations, we defined a set of non-prehensile actions for the robot, to mimic the natural human behaviour.

Fingers names are defined p, i, m, a, x for thumb, index, middle, ring, little, respectively (see fig. 12).

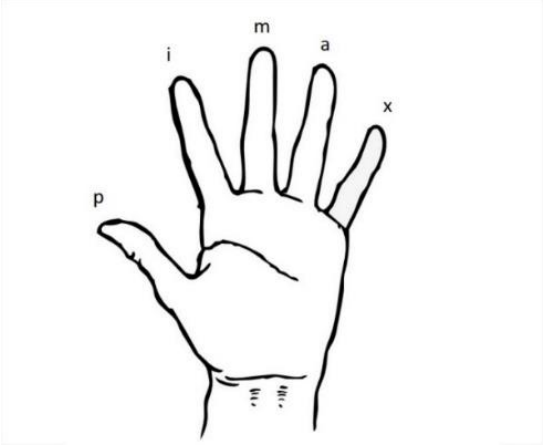

Figure 12: Hand with finger names

2. Defined actions

To understand how humans, manipulate deformable materials, Cherubini et al. (2019) ran a pilot user study with 9 volunteers (age range: 20-40; 6 male, 3 female). Each participant is asked to shape kinetic sand in a sandbox, while being recorded with a fixed RGBD camera (Intel RealSense SR300, resolution 640 × 480). The database is publicly available (see Cherubini et al. (2019)).

We used this database to observe the natural actions of humans and extract the more useful ones to define out actions. As a result, 5 actions are defined as poke, tap, grasp, drag, pinch.

a) Poke

Poke action describes pushing towards the sand with i (see fig. 13).

Figure 13: Poke action

b) Tap

Tap action describes pushing towards the sand with an open palm (see fig. 14).

Figure 14: Tap action

c) Grasp

Grasp action describes first, opening the hand shaped as a claw then, gathering the material (see fig. 15).

Figure 15: Grasp action

d) Drag

Drag action describes moving the material with the hand shaped as a closed claw while the hand is aligned with the desired direction (see fig. 16).

Figure 16: Drag action

Drag has an additional pan angle parameter which describes the direction to move the sand. Eight directions are defined which are: north (N), northeast (NE), northwest (NW), east (E), west (W), south (S), southeast (SE), southwest (SW), as on a compass (see fig. 17).

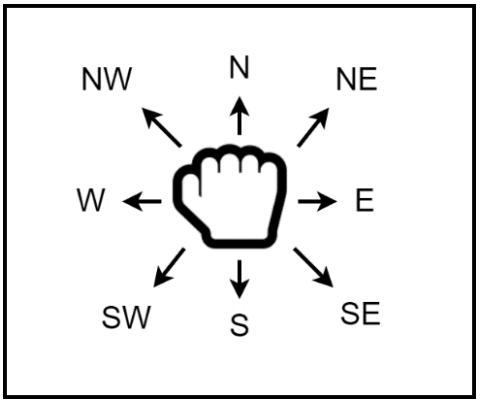

Figure 17: Drag action pan angle parameter

e) Pinch

Pinch action describes pinching the material between p and i (see fig. 18).

Figure 18: Pinch action

Pinch has an additional tilt angle parameter which describes the angle between the manipulation area and the hand direction (see fig. 19). Three angles are defined which are 0°, 30° and 60°.

Figure 19: Pinch action with tilt parameter γ

3. Action generation

Actions are performed using a humanoid robot hand attached to the tip of a robot arm to imitate real human limbs.

a) Environment

A rectangular shaped manipulation area is divided by rectangle grids. These small rectangles define the locations where the action will take place. These rectangles will be called grid for the rest of the text.

When the hand is in contact with the material for the first time, the center point of the palm of the hand C, should be aligned with the center point of a grid (see fig.26).

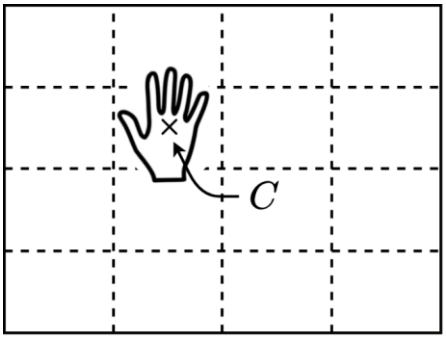

Figure 20: Robot hand on the manipulation area on 4x4 grid and C the center point of the palm of the hand.

b) Robot arm

The arm is modelled by a 7-axis robot arm used to mimic a real human arm. The home, begin, end poses of the arm are calculated in advance. The robot is controlled by an inverse kinematics solver using these coordinates.

c) Robot hand

A robot hand is attached at the end of the robot arm to perform the actions. Since the robot hand has too many axis, it is controlled by forward kinematics. Joint positions are obtained after manually setting the hand to the desired shape.

d) Control architecture

To perform an action, both arm and hand should change their pose. Let's call the position, orientation of the arm, hand, Px, Ox and define a home, begin, end poses of the arm, hand, Home_x, Begin_x, End_x, both with x: a, h as arm, hand.

Home_x defines the default pose of the robot. Every action starts and ends at Home_x.

Begin_x and End_x are the beginning and ending pose of an action. Every action has different Begin_x and End_x. For some actions, since the pose of the hand is the same from the beginning to the end, Begin_h and End_h are the same.

The robot is controlled in the following order:

Oa @ Home_a -> Oh @ Home_h -> Pa @ Home_a -> Ph @ Home_h -> Oa @ Begin_a -> Pa @ Begin_a -> Oh @ Begin_h -> Ph @ Begin_h -> Oh @ End_h -> Ph @ End_h -> Oh @ Begin_h -> Ph @ Begin_h -> Oa @ End_a -> Pa @ End_a -> Pa @ Home_a -> Oa @ Home_a -> Ph @ Home_h -> Oh @ Home_h

This architecture ensures the beginning and end pose to be at Home_x and prevents unintended collisions with the manipulation area. See fig. 27 for examples of Home_x, Begin_x, End_x of an action.

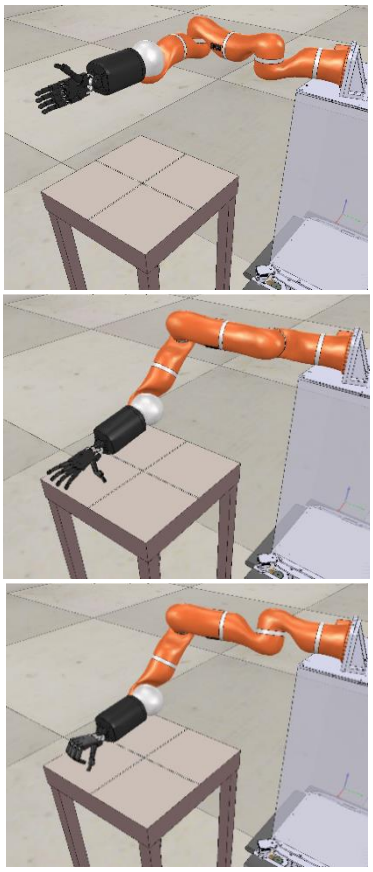

Figure 21: Example of Home (top), Begin (center), End (bottom) poses for both arm and hand. The modelled action is the Grasp.

4. Image prediction

The output shape of the object after performing an action for a given input shape is predicted by an artificial neural network (ANN). Since the actions only affect the shape, the difference can be calculated by comparing the 2D depth map of two poses.

a) Depth camera

2D depth images are produced by a depth camera placed at the top of the object's enclosure with the optical axis perpendicular to it.

The luminance I_{uv} of pixel u, v of the depth camera of resolution (w, h) and quantification b bits is:

$$I_{uv} = [0, 2^b - 1] \subset \mathbb{N} \qquad eq. 3$$

Assuming a linear depth model for any point in the image, the luminance can be converted to the point depth Z (in meters) in the camera frame.

$$Z = \frac{Z_{min} - Z\_max}{2^b - 1} I_{uv} + Z_{max} \in [Z_{min}, Z_{max}] \subset \mathbb{R}^+ \qquad eq.4$$

This requires a prior knowledge about the camera depth range. It is important to note that this model clips at the maximum, minimum depth of all points which in reality have more, less depth. Assuming undistorted perspective projection, the two other camera frame coordinates are:

$$X = \frac{u - u_0}{f_x} Z \quad and \quad Y = \frac{v - v_0}{f_y} Z \qquad eq.5$$

with $(u_0, v_0) \in [1, \mathrm{w}] \times [1, h]$ the camera principal point coordinates in the image plane, (w, h) are image width and heights and (f_x, f_y) the camera focal lengths expressed in pixels.

Defining $x = \frac{u-u_0}{f_x}$ and $y = \frac{v-v_0}{f_y}$, equation becomes:

$$X = xZ \quad and \quad Y = yZ \qquad eq.6$$

b) Region of interest

The begin (i.e. shape of the object before performing the action) and the end (i.e. shape of the object after performing the action) images are annotated according to the bounding box which contains a difference in depth. We call these regions the region of interests (ROIs). For each action, the images are cropped according to the mean average of the surface areas of the ROIs on the image plane. This step excludes the unnecessary data and helps to reduce the computational resources as well as the time required to train the ANN.

ROIs are defined by $(x_{TL}, y_{TL}, x_{BR}, y_{BR})$ as x, y are coordinates in image, TL, BR are top-left, bottom-right corner of the ROI. After extracting ROIs for a set of data, the mean averages of ROI (see fig. 19), $ROI^m = (x_{TL}^m, y_{TL}^m, x_{BR}^m, y_{BR}^m)$ is calculated by:

$$C^{ROI} = \left(\frac{x_{BR} - x_{TL}}{2}, \frac{y_{BR} - y_{TL}}{2}\right) \qquad eq.7$$

$$W^{ROI} = \left(\frac{\sum_n x_{BR} - x_{TL}}{n}, \frac{\sum_n y_{BR} - y_{TL}}{n}\right) \qquad eq.8$$

$$ROI^m = \left(C_x^{ROI} - \frac{W_x^{ROI}}{2}, C_y^{ROI} - \frac{W_y^{ROI}}{2}, C_x^{ROI} + \frac{W_x^{ROI}}{2}, C_y^{ROI} + \frac{W_y^{ROI}}{2}\right) \qquad eq.9$$

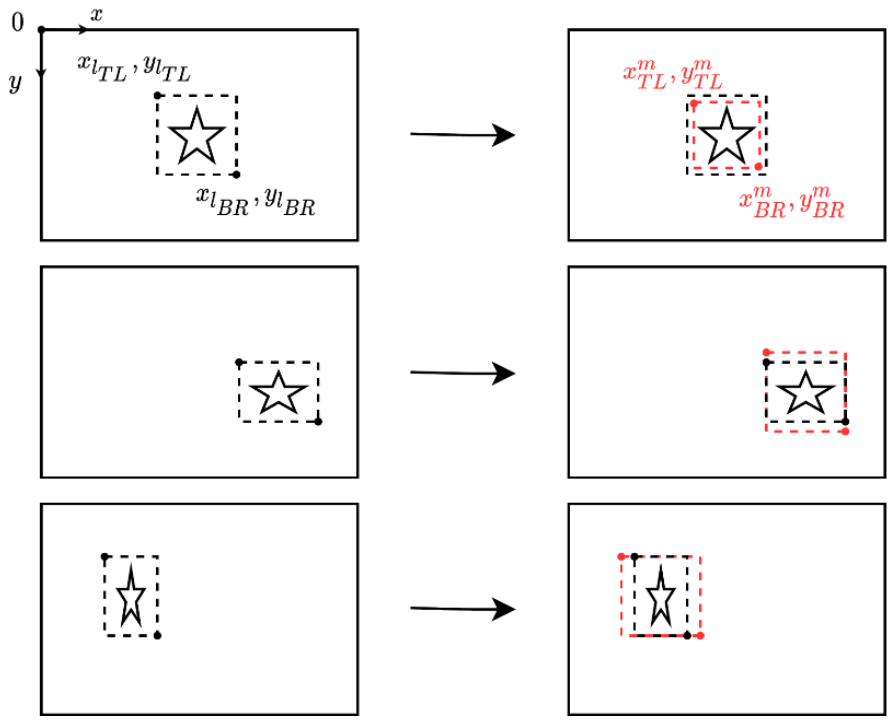

Figure 22: Three poses of an action on the image plane. The effect of the action is represented with star. Black dashed lines represent the ROI for that pose. Red dashed lines are the ROI^m of the action.

c) Distance

Finding the distance between the begin and end ROI is a necessary step for both the ANN and decision tree. ANN is using the distance between the output and ground truth as a loss function. Decision tree is using it to find the shortest path that leads to the desired shape.

Using the previous equations in chapter: depth camera, the relative distance between p, q the same pixel in images I_p and I_q can be calculated as:

$$\|p - q\| = \sqrt{\left(xZ_p - xZ_q\right)^2 + \left(yZ_p - yZ_q\right)^2 + \left(Z_p - Z_q\right)^2} \qquad eq.10$$

which can be expressed as:

$$\left|\|p - q\|\right| = R_{uv}\frac{Z_{max} - Z_{min}}{2^b - 1}\left|I^p_{uv} - I^q_{uv}\right| \qquad eq.11$$

with $R_{uv} = \sqrt{1 + x^2 + y^2}$ and I^p_{uv}, I^q_{uv} the luminance of pixel (u, v) in image I_p, I_q. The value R_{uv} is constant on ellipses centred at the image principal point (where it is equal to 1) and it increases with the sizes of the ellipses (see fig. 20).

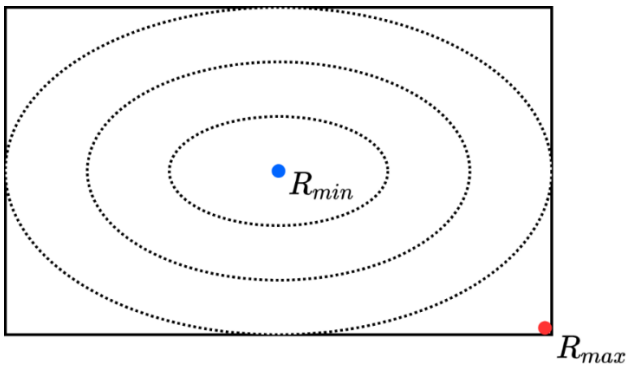

Figure 23: Representation of R values on image

Using eq.10 and 11, distance d between two point in image I_p, I_q can be expressed as:

$$d(I_p, I_q) = \frac{\sum_{(u,v)} \sum_{(u,v)} R_{uv} |I_{uv}^p - I_{uv}^q|}{wh(2^b - 1)R_{max}} \qquad eq.12$$

which highly resembles the mean absolute error loss function, commonly used in ANNs, with the addition if the R_{uv} weights normalized by R_{max}.

From the same logic, weighted and normalized (between 0,1) image $I^{\#}$ can be denoted as:

$$I^{\#} = \frac{R \times I}{wh(2^b - 1)R_{max}} \qquad eq.13$$

d) Data augmentation

ANNs require a significant amount of data due to their nature. To provide this, we artificially augment the number of images that we have.

- Functions

Images are augmented using following functions: rotation, horizontal flip, vertical flip, zoom, translation, brightness, contrast, gaussian noise, salt and pepper noise, poisson noise, speckle noise. See fig. 21 to view their effects.

Rotation function is rotating the image around the center of the image. Zoom function is zooming the image towards the center.

For the zoom and translation, black paddings are added if necessary, to get an image of the same size.

Figure 24: From top-left to bottom-right: Original image, rotation, horizontal flip, vertical flip, zoom, translation, brightness, contrast, gaussian noise, salt and pepper noise, poisson noise, speckle noise.

- Pipeline

The image is passed through a pipeline consist of augmentation functions below. Every function has a probability $p_a = [1, 100] \in \mathbb{R}^+$ for the activation. For each pass, a random number $f_a = [1, 100] \in \mathbb{R}^+$ is generated. The function is activated if $f_a \leq p_a$.

For example, if the activation probability is 50, the numbers from 1 to 50 activates and from 51 to 100 deactivates the function.

For each parameters of the function (if any), a range is defined. For each pass, a random number from the range is generated and set.

This pipeline architecture ensures random generated images (see fig. 22).

The goal is to augment images in begin-end pairs to prevent alternating the mapping between the pairs. As a result, the same augmentation is applied both to the begin and the end image of the same pose.

Figure 25: Original (left) and augmented (right) image

e) Autoencoder

An autoencoder is used to predict the resulting shape after applying an action to a given input shape. An input shape in form of a depth image is fed to the network, then the network outputs a predicted depth image after applying the action.

- Data preperation

Images are classed according to their <action>/<instance>/<pose> with action as the name of the primitive action, instance as the begin or the end image, pose as the number of samples. For example: action: poke/ instance: begin/ pose: no.3

- Architecture

The autoencoder model is a combination of 2 networks. An encoder network which takes an image as input and outputs its latent space representation, is followed by a decoder network which has an input and an output in the reverse order (see fig. 23). The complete autoencoder takes an image as input, generates another one at the end.

The encoder consists of 4 convolutional layers followed by 2 fully connected (FC) layers with a maxpool layer after each convolution. The convolution layers act as feature extractor while the FC and maxpool layers work on the dimensionality reduction. Since the ROI size of the actions are not consistent, the input dimensions of the network are variable. This allows to produce an appropriate number of trainable weights which leads images of similar quality for every action. The leaky ReLu activation function with $a = 0.2$ is used across the network.

The convolution layers are constructed as *kh x kv x o* with *kh, kv* as the horizontal, vertical kernel size and o as the number of output channels. In the encoder network, they are shaped respectively *5 x 5 x 32, 5 x 5 x 64, 3 x 3 x 128, 1 x 1 x 256*. All the maxpool layers have *kh x kv = 2 x 2*.

The decoder network is symmetrical to the encoder. To reverse the effects of convolution and maxpool layers, they are swapped by transposed convolution (i.e. deconvolution) and upsampling layers while preserving the same *kh, kv, o*. Since the dimension of the data after the last convolution layer depends on the input, the data is reshaped to the same dimension before feeded to the first deconvolution layer. The ReLu activation function is used across the decoder network.

The complete autoencoder architecture used in this project is illustrated in fig. 24.

Both on the encoder and decoder networks, strides and paddings for the convolution, deconvolution and maxpool layers are set to 1 and half padding, respectively. Eq. 15 shows the calculation of output width, height w_{out}, h_{out} for a given input width, height w_{in}, h_{in} for the half padding with horizontal and vertical strides as s_w, s_h.

$$w_{out} = \frac{ceil(w_{in})}{s_w} \quad and \quad h_{out} = \frac{ceil(h_{in})}{s_h} \qquad eq.14$$

Padding may cause the output to be bigger than the input. Since the padding expands the image from borders, output is cropped to same dimensions as the input.

- Loss function

Since the objective is to get a prediction of the end image from a given begin image, the loss function is constructed between the prediction generated by the autoencoder and the end pair of the inputted begin image. During the training phase of the autoencoder, the loss function should minimize the distance between these two images. To achieve that, the previously shown distance formula in eq.13 is defined as loss function (see eq. 16).

$$loss = d\left(I^{+}, I_{pred}\right) \qquad eq.15$$

with I^{+} as the ground truth of the end image pair of I and I_{pred} as the predicted end image of the image I.

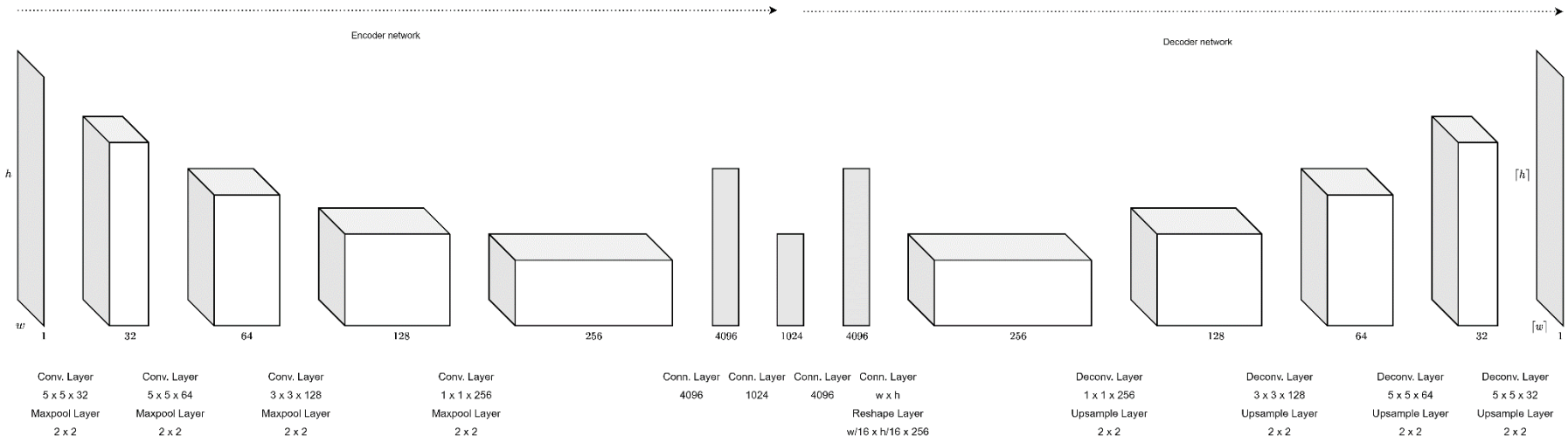

Figure 26: Autoencoder architecture. Conv., conv. represent convolution and fully connected layers. Numbers below layers represent the number of filters for the conv. and number of neurons for the conn.. Conv. layers are followed by a maxpool layer with parameters as specified.

5. Sequence selection

The shortest sequence of actions to get the desired shape from an initial shape is found by creating a tree of all possible action combinations, then comparing the end results and the desired shape by previously viewed distance metric.

a) Decision tree

A decision tree is generated from all possible action combinations. Then, the shortest path $s_* = \{a_1, \dots, a_k\}$ which leads from the initial to the desired image is found using the algorithm in fig. 25.

Algorithm 1: Action sequence generator

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

Initialize $k = 0$, $\mathbf{s} = \emptyset$
obtain $\mathbf{I}^{\#}_{0|\emptyset}$ from $\mathbf{I}_0$ and $\mathbf{I}^{\#}_{*}$ from $\mathbf{I}_*$
loop
  for each image $\mathbf{I}^{\#}_{k|\mathbf{s}}$ **do**
    for each action $\mathbf{a}^{ij}$ **do**
      extract submatrix $\underline{\mathbf{I}}^{ij}$ from $\mathbf{I}^{\#}_{k|\mathbf{s}}$
      predict effect of $\mathbf{a}^{ij}$ on $\underline{\mathbf{I}}^{ij}$: $\underline{\mathbf{I}}^{\mathbf{a}^{ij}}$
      inject $\underline{\mathbf{I}}^{\mathbf{a}^{ij}}$ in $\mathbf{I}^{\#}_{k|\mathbf{s}}$ to get $\mathbf{I}^{\#}_{k+1|\{\mathbf{s},\mathbf{a}^{ij}\}}$
      update sequence $\mathbf{s} = \{\mathbf{s}, \mathbf{a}^{ij}\}$
    end for
  end for
  for each image $\mathbf{I}^{\#}_{k+1|\mathbf{s}}$ **do**
    compute $d_{\mathbf{s}} = d\left(\mathbf{I}^{\#}_{k+1|\mathbf{s}}, \mathbf{I}^{\#}_{*}\right)$
  end for
  if $\exists\, d_{\mathbf{s}} < \bar{d}$ **then**
    return $\mathbf{s}_*$ which yields the lowest $d_{\mathbf{s}}$
  end if
  $k = k + 1$
  if $(k \mod k_{max}) \neq 0$ **then**
    delete all $\mathbf{I}^{\#}_{k|\mathbf{s}}$ images except the p with lowest $d_{\mathbf{s}}$
  end if
end loop

Figure 27: Decision tree algorithm

For each possible action at the grid location i,j, a^{ij}, a map is required from prior image $I^{\#}_k$ to posterior (after execution of a^{ij}) image $I^{\#}_{k|a^{ij}}$. Within a set of candidate sequences of actions S each leading from $I^{\#}_0$ to a different image $I^{\#}(s)$, the one leading to the image that is the closest to the $I^{\#}_*$ will be chosen:

$$s_* = argmin_{s \in S}\ d(I^{\#}(s), I^{\#}_*) \qquad eq.16$$

This sequence corresponds to the shortest path in the decision tree of all possible images generated by all possible actions A.

The number of images at layer k can be calculated using eq.17.

$$n_{img} = (\dim(A) * grid_h * grid_v)^k \qquad eq.17$$

with $grid_h, grid_v$ as the horizontal and vertical grid sizes.

During training phase, the autoencoder is set minimize the distance d between its output and the ground truth.

Once the autoencoder has been trained, and a reliable posterior submatrix $\overline{I_{k|a}}$ can be outputted. Using the output information, the corresponding submatrix in $I_k^{\#}$ is replaced by $\overline{I_{k|a}}$ to obtain $I_{k|a,T}^{\#}$.

For each action and pose a^{ij}, the mapping is obtained in the following way:

1) Generate a tree from $I_0^{\#}$ to all possible (i.e. obtained after executing any action in A) posterior images $I_1^{\#}$.
2) If any of the posterior images is near enough (has $d < \bar{d}$) to $I_*^{\#}$, return the assicuated sequence s_*.
3) Otherwise, move to the next iteration k, by propagating each $I_k^{\#}$ to all possible posterior images $I_{k+1}^{\#}$.
4) Since the tree will grow very quickly ($(\dim A)^k$ leaves at iteration k), every k_{max} iterations prune it by removing all the images, except those along the sequences leading to the best p (in terms of d)

VI. Experimental Setup

1. Depth camera

Depth images are obtained by an Intel RealSense Depth Camera D435 (see fig. 28). According to the datasheet (see Intel RealSense D400 Series Product Family Datasheet), the nominal values the camera parameters are: $u_0 = 424, v_0 = 240, f_x = 415, f_y = 373$.

Dimensions of the obtained images are 848×480 as width x height respectively.

Using these values, the R values are calculated between $[1, 1.568]$.

Figure 28: Intel RealSense D435

2. Training setup

The autoencoder is trained under Google Colab environment with GPU and high RAM settings (see Google Colab FAQ).

Hardware specifications given by Google are:

- CPU: Intel Xeon @ 2.30GHz
- GPU: Nvidia Tesla P100 PCIE-16Gb
- RAM: 26.30 Gb

3. Simulator

The method was first tested in the CoppeliaSim robot simulator by Coppelia Robotics before moving on to the real-world experiment.

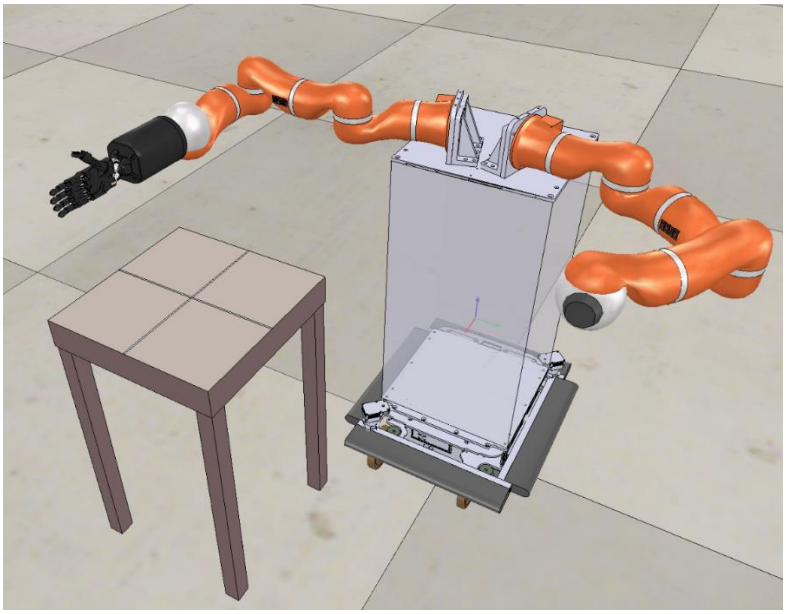

Figure 29: CoppeliaSim scene

The scene consists of two robot arms attached to a mobile base and a table. Robot hands are attached at the tip of both robot arms (see fig.29).

For simplicity, the mobile base is considered fixed and only the right robot arm with hand is used during the simulation.

The robot arm used on the simulation is a clone of LWR by KUKA. The hands are clones of Shadow Dexterous Hand by Shadow Robot Company. Both the arm and hand are controlled by the inverse kinematics solver of the simulator. Desired poses are set manually, a finite state machine is coded to follow the desired control architecture.

The table represents the manipulation area. For simplicity, it is divided into 2x2 equal rectangles instead of 4x4 from the original method.

4.KUKA light weight robot

The Kuka-DLR Lightweight Robot (LWR) is a 7 degrees-of-freedom (DOF) robot arm designed to imitate a human arm while being safe to operate around humans (see fig. 30). It has the human arm's dexterity, sensing and strength. It is also less dangerous and easier to program than other robots, making it ideal for tasks that require close human-robot interaction.

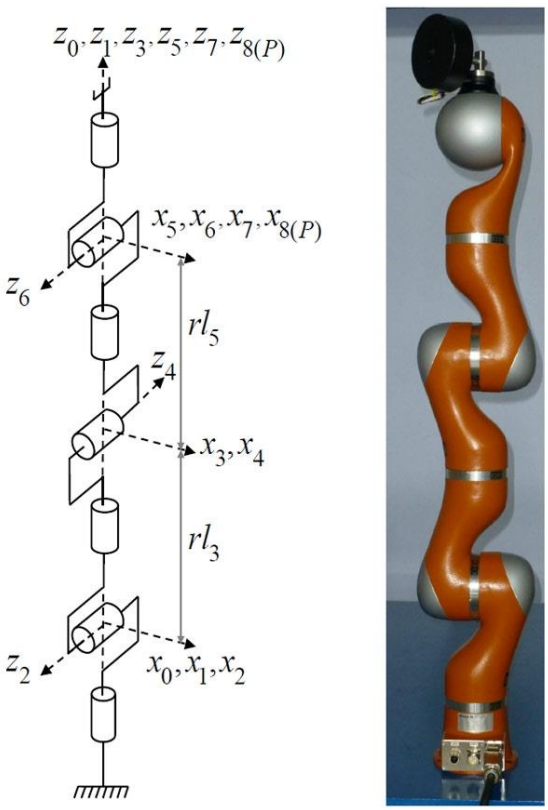

Figure 30: Kuka LWR and its articulations

Technical details of the KUKA LWR are presented below (see Bischoff et al. (2010)):

- Payload: 7 kg
- Number of axes: 7
- Mounting position: Any
- Repeatability (ISO 9283): +- 0.05 mm
- Weight (excluding controller), approx.: 16 kg

5.Shadow Dexterous Hand

The Shadow Dexterous Hand is a humanoid robot hand system that provides 24 movements to reproduce as closely as possible the kinematics and dexterity of the human hand (see). It is designed to provide comparable force output and movement precision to the human hand.

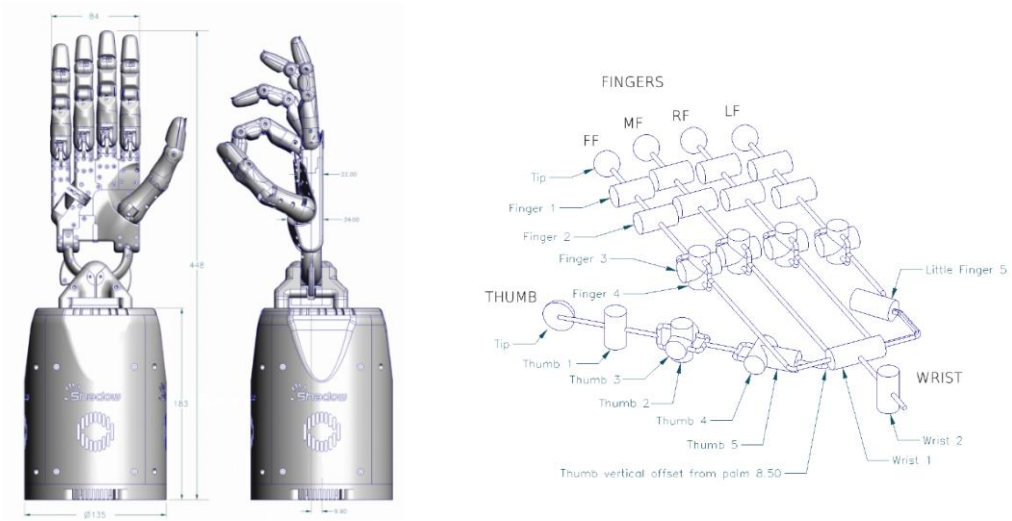

Figure 31: Shadow Dexterous Hand dimensions (left) and its articulations (right)

It is measured to be similar in shape and size to a typical male hand. The fingers are all the same length, with the knuckles staggered to give comparable fingertip locations to the human hand.

The hand has 24 joints. It has 20 DOF, greater than that of a human hand. The thumb has 5 DOF and 5 joints. Each finger has 3 DOF and 4 joints (see fig. 31). The Hand and forearm have a total weight of 4.3 kg. The Hand, while in a power-grasp, can hold up to 5 Kg (see Shadow Dexterous Hand Technical Specification).

6. Real-world setup

Real-world setup consists of a Shadow Dexterous Hand attached to the tip of a KUKA LWR arm. Intel RealSense D435 depth camera is placed at the top of the manipulation area. Kinetic sand is used as a deformable material (see fig. 32).

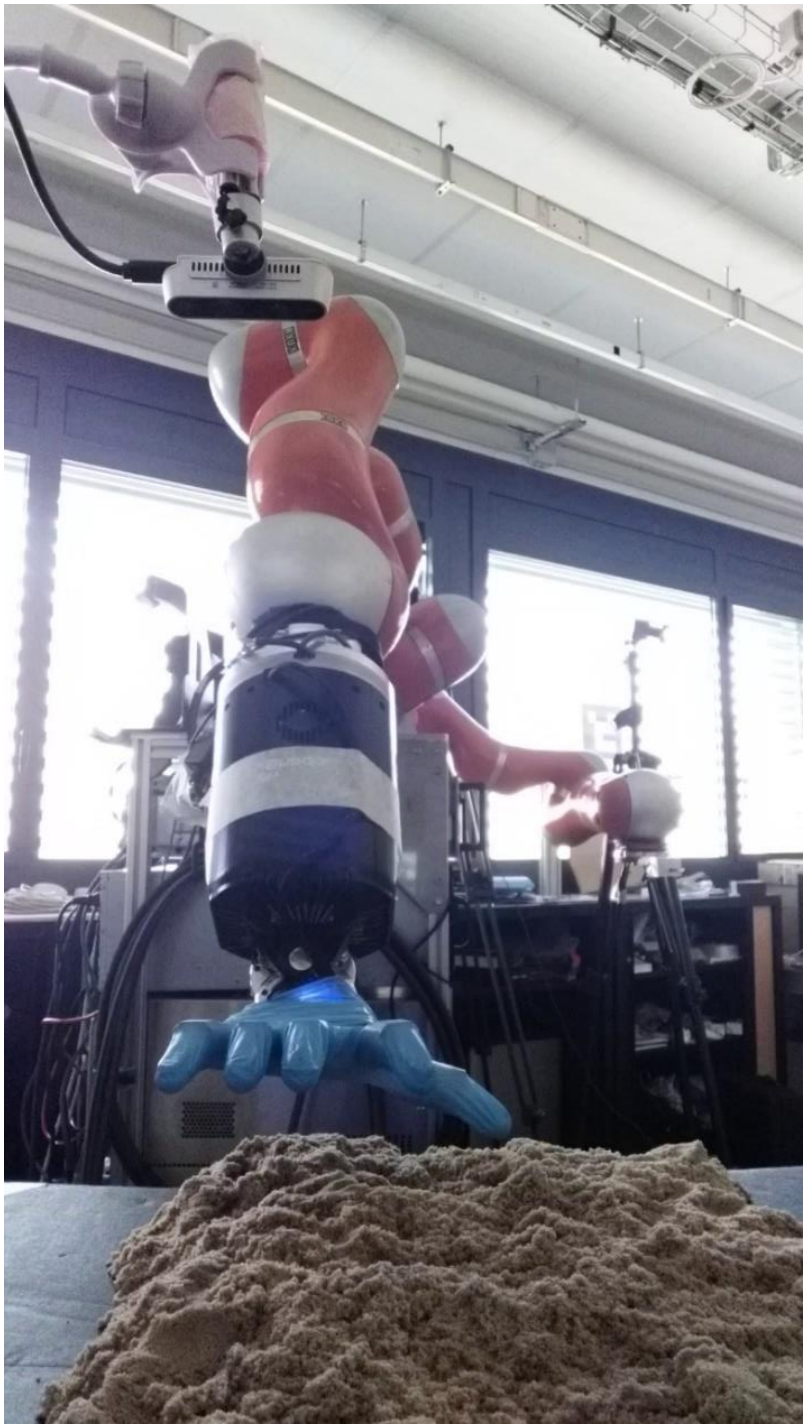

Figure 32: Real-world setup of the experiment at LIRM

a) Kinetic sand

Kinetic sand is used as a deformable plastic material. Originally, it is a toy material which mimics the physical properties of wet sand. It is made out of 98% regular sand and 2% polydimethylsiloxane (a viscoelastic silicone) and it can be molded into any desired shape. It does not stick to any materials other than itself and does not dry out.

a) RKCL framework

The robot kinematic control library (RKCL) framework is used to control the robot arm and hand (see rkcl-framework website). RKCL is a free and open-source software dedicated to the control of robots. It is organized as a framework of packages containing software libraries and tools written in C++. RKCL has been made generic to be compatible with any robot characterized by a kinematic tree structure. A custom package for the project is generated for our project.

The major benefit of RKCL is its ability to deal with multiple robots in the same control process. The control strategy takes into account synchronization issues to minimize the error arising from using different hardware components. This feature is important since our method consists of two robots which are the arm and hand.

VII. Results

The experiments are only performed on the poke and tap action. Because of the limited time, the tap action is not trained properly on the autoencoder, hence only the poke action will be presented in this section.

There are 4 main steps to be followed:

1) Raw depth images coming from data is manually annotated to get the ROIs.
2) The average ROI for each action is calculated and cropped accordingly.
3) Autoencoder is trained using cropped images.
4) Decision tree gives the best sequence of actions with minimum distance metric and its result.

1.Depth images

A video is recorded by the depth camera while the actions are performing using human hands. For the poke action, 10 different poses are recorded. Then, the video is separated into images (see fig. 33). Only the images which shows the full effect of the action and do not contain the manipulation tools (hand in this case) are preserved. Images are manually annotated according to their ROI.

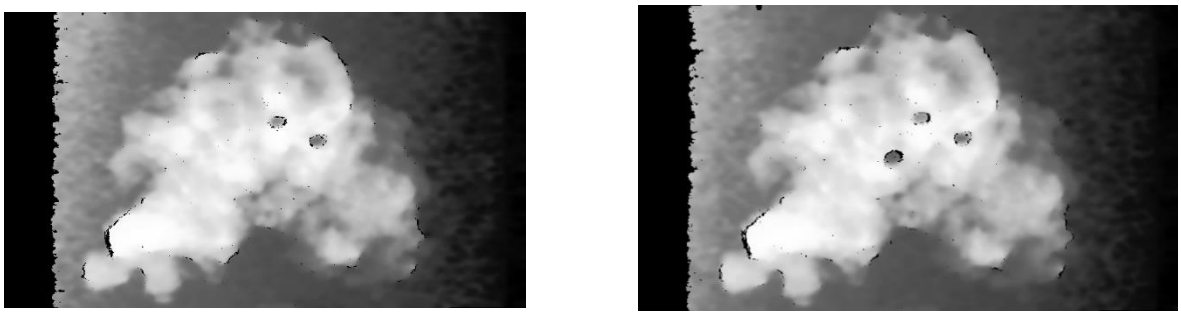

Figure 33: Raw images extracted from the video footage which contain the full effect of the poke action.

2.Data preparation

The mean average of the ROIs, ROI^m, is calculated using previously shown equation. Then, raw images are cropped according to ROI^m. Images are classified as begin and end instances. Begin, end instance of a given pose represents the image before, after applying the action.

Cropped and classified images are artificially augmented (see fig. 34). In our experiment, each pose is augmented to get 100 images per pose. Since we had 10 poses for poke, the number of total images become 1,000.

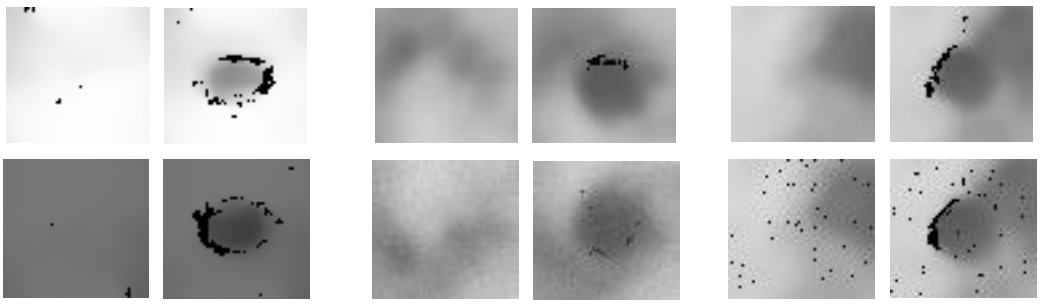

Figure 34: Cropped and augmented images. For each group of 4, top row, has the begin (left) and end (right) instance images, bottom row has an example of their augmented versions

Image augmentation parameters p_a for each function and their ranges (if any) are set as presented below:

- Rotation: $p_a = 50$, range = [-150°, +150°]
- Horizontal flip: $p_a = 50$
- Vertical flip : $p_a = 50$
- Zoom : $p_a = 33$, range =]%0, %150]
- Translation: p_a =33, range towards left, right, top, bottom =]%0, %10]
- Brightness: $p_a = 25$, range = [-%50, +%50]
- Contrast: $p_a = 25$, range = [%10, %110]
- Gaussian noise: $p_a = 16$, range of mean =]0, 3], of variation =]0, 1]
- Salt pepper noise: $p_a = 16$, range of ratio = [0 ,1], of amount =]%0, %0.05]
- Poisson noise : $p_a = 16$, range of variation =]0, 50]
- Speckle noise: $p_a = 10$, range of mean =]0, 2], of variation =]0, 1]

3. Autoencoder

Autoencoder is trained by %70 of the available data for that action. The remaining %30 is used for testing. Since poke has 10 poses and 100 images for each pose (after data augmentation), 70 images are randomly chosen in each pose to be used in training.

Adam optimizer with the following parameters is used during training: $\alpha = 0.0002$, $\beta_1 = 0.5$, $\beta_2 = 0.999$, $\epsilon = 1e-07$

For the poke, using 7,000 images in total, autoencoder is trained for 500 epochs in about 50 minutes. During training, end image is used as ground truth. The prediction is obtained directly from the begin image. Fig. 35 shows the evolution of loss between the ground truth and the prediction. Since the learning curve is flattened after 250 epoch, training can be stopped earlier to reduce the time. Results of the training can be seen in fig. 36.

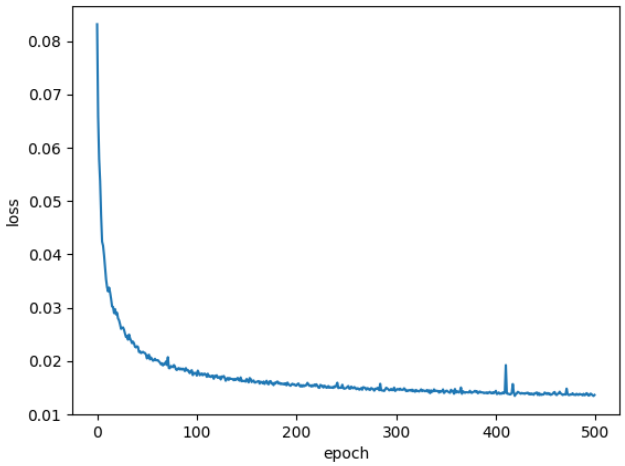

Figure 35: The learning curve of autoencoder

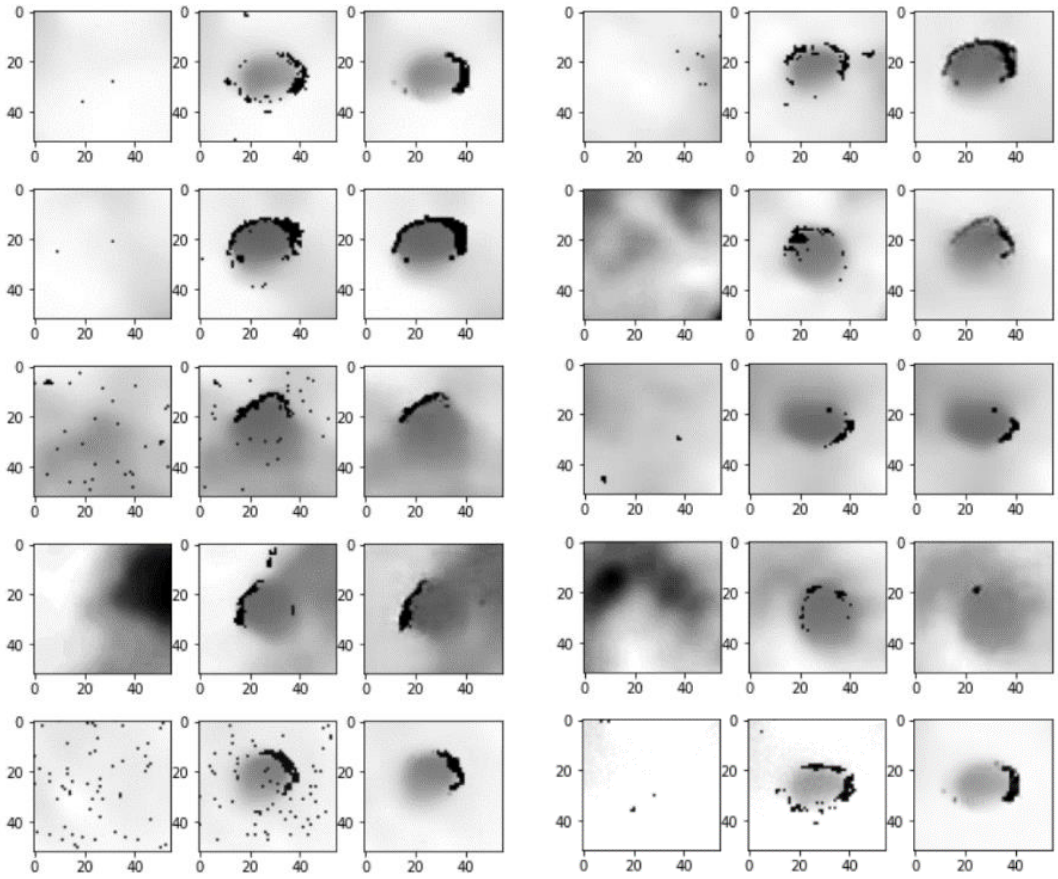

Figure 36: Begin (left) and end (right) image pairs and autoencoder's output (right) of an action. One image per each pose's augmented images is presented.

4. Decision tree

Decision tree algorithm is initiated with possible actions $A = (poke, tap)$, $\bar{d} = 0.0005$, $k_{max} = 3$, 2x2 and 4x4 location grids. 2x2 grid is preferred when the speed is more important than the precision. It is important to note that, without pruning, using 4x4 grid, the total amount of images at the final layer calculated by eq. 17 is 32,768 images instead of 512 images using 2x2 grid.

In fig. 37, algorithm is executed with 2x2 grid to achieve the closest shape as distance metric. The sequence $s_* = (poke^{2,1}, poke^{1,2}, poke^{2,2})$ is found with distance $d = 44e - 6$. In fig. 38, grid dimensions are changed to 4x4. Outputted result is the sequence $s_* = (poke^{2,2}, poke^{2,3}, poke^{3,2})$ with the distance $d = 50e - 6$. In these two examples, actions are intentionally performed close to the grid center points. As a result, the correct sequence is found with a low d value.

In fig. 39, actions are intentionally performed close to the edge of grid rectangles. A sequence with $d < \bar{d}$ can not be found. The sequence which has the smallest d value is found $s_* = (poke^{2,2}, poke^{2,2}, poke^{2,2})$ with $d = 0.0013$. Since the decision tree algorithm considers the actions to be performed at the center of the grid, correct sequence could not be found.

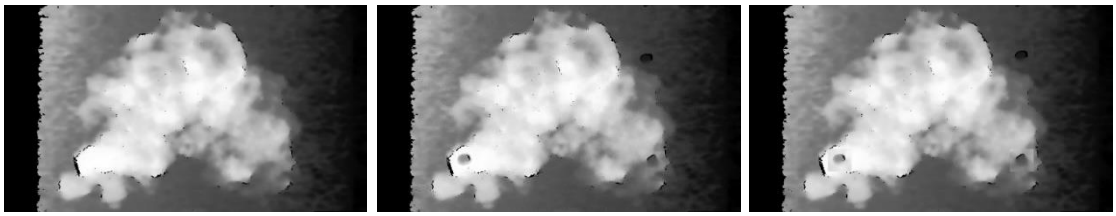

Figure 37: Initial (left) and desired (right) shape with 2x2 grid. Algorithm outputs $s_* = (poke^{2,1}, poke^{1,2}, poke^{2,2})$ *with* $d = 44e - 6$

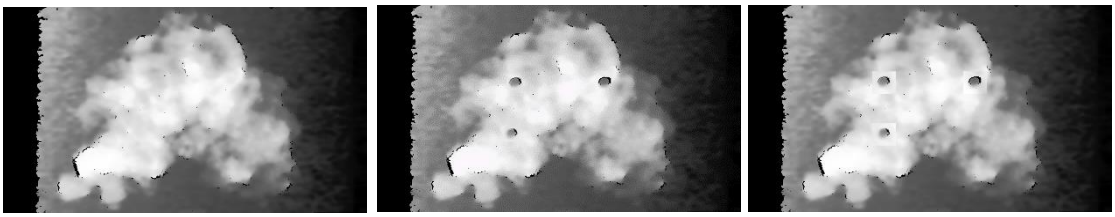

Figure 38: Initial (left) and desired (right) shape with 4x4 grid. Algorithm outputs $s_* = (poke^{2,2}, poke^{2,3}, poke^{3,2})$ with $d = 50e - 6$

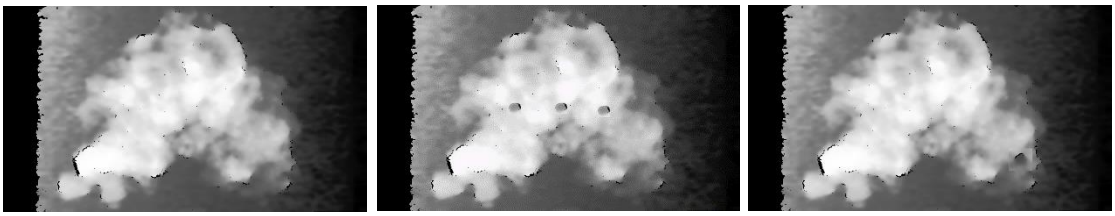

Figure 39: Initial (left) and desired (right) shape with 2x2 grid. Algorithm outputs $s_* = (poke^{2,2}, poke^{2,2}, poke^{2,2})$ with $d = 0.0013$.

VIII. Conclusion

In this project, we have addressed the problem of shaping of deformable materials. Five actions are defined to manipulate the material by using real human observations: poke, tap, grasp, drag, pinch. Actions are first tested on simulator, then, implemented in real-world robots.

The shape in form of depth images is obtained by a depth camera. A distance metric is defined to express the distance difference between two images.

An autoencoder model is trained using distance metric to predict the effects of actions to a given input shape.

The closest shape possible shape to the desired one is searched using the distance metric after applying all possible combination of actions to the initial shape using a decision tree algorithm.

Our method is highly applicable in to the real-word. Unfortunately, due to time limitations, only poke and tap actions are shown in real-world results.

One possible limitation of our method could be the computational requirement with bigger number of grid size. The manipulation area should be divided in to a grid size which is enough to produce actions at accurate positions. This also impose a limitation on the distance metric. If the desired shape has a deformation close at the edge of the grid rectangle, the decision tree may not be able to find the correct sequence. The action will not be performed on that location; hence, the distance metric will be high.

The search of finding the shortest path which leads to the desired shape corresponds in the decision tree of all possible images generated by all possible actions. This is a classical motion planning problem, which can be solved using many state of art algorithms (A$*$, RRT, etc). This part of the project is developed to work as simple as possible. We leave to path planning experts the pleasure of proposing the best among such algorithms.

All in all, the results show that our method succeeds in making the robot achieve the desired shape. Nevertheless, we acknowledge the limitations of our exploratory work. These could be the object of future work of researchers interested by this fascinating topic.

IX. References

Arriola-Rios VE and Wyatt JL (2017) A multi-modal model of object deformation under robotic pushing. IEEE Transactions on Cognitive and Developmental Systems 9(2): 153–169.

Berenson D (2013) Manipulation of deformable objects without modeling and simulating deformation. In: IEEE/RSJ Int. Conf. on Robots and Intelligent Systems, IROS.

Bischoff R , Kurth J, Schreiber G, Koeppe R, Albu-Schäffer A, Beyer A, Eiberger O, Haddadin S, Stemmer A, Grunwald G, Hirzinger G (2010): The KUKA-DLR Lightweight Robot arm – a new reference platform for robotics research and manufacturing. ISR/ROBOTIK

Cherubini A, Leitner J, Ortenzi V and Corke P (2018) Towards vision-based manipulation of plastic materials. In: IEEE/RSJ Int. Conf. on Robots and Intelligent Systems, IROS.

Cherubini A, Ortenzi V, Cosgun A, Lee R, Corke P (2019) Model-free vision-based shaping of deformable plastic materials. In: The International Journal of Robotics Research, 0278364920907684

Cretu AM, Payeur P and Petriu EM (2012) Soft object deformation monitoring and learning for model-based robotic hand manipulation. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42(3): 740–753

Das J and Sarkar N (2011) Autonomous shape control of a deformable object by multiple manipulators. Journal of Intelligent and Robotic Systems 62(1): 3–27.

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

Dumoulin V, Visin F (2016): A guide to convolution arithmetic for deep learning. arXiv preprint arXiv: 1603.07285

Elliott S and Cakmak M (2018) Robotic cleaning through dirt rearrangement planning with learned transition models. In: IEEE Int. Conf. on Robotics and Automation, ICRA.

Ficuciello F, Migliozzi A, Coevoet E, Petit A and Duriez C (2018) FEM-based deformation control for dexterous manipulation of 3D soft objects. In: IEEE/RSJ Int. Conf. on Robots and Intelligent Systems, IROS.

Gemici MC and Saxena A (2014) Learning haptic representation for manipulating deformable food objects. In: IEEE/RSJ Int. Conf. on Robots and Intelligent Systems, IROS.

Gopalakrishnan K and Goldberg K (2005) D-space and deform closure grasps of deformable parts. The International Journal of Robotics Research 24(11): 899–910.

Higashimori M, Yoshimoto K and Kaneko M (2010) Active shaping of an unknown rheological object based on deformation decomposition into elasticity and plasticity. In: IEEE Int. Conf. on Robotics and Automation, ICRA.

Howard AM and Bekey GA (2000) Intelligent learning for deformable object manipulation. Autonomous Robots 9(1): 51–58.

Isele D, Rahimi R, Cosgun A, Subramanian K and Fujimura K (2018) Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 2034–2039.

Li FF, Krishna R, Xu D, Byun A: CS231n (2020): Convolutional Neural Networks for Visual Recognition. Stanford Vision and Learning Lab

Li Y, Wu J, Tedrake R, Tenenbaum J and Torralba A (2018) Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. arXiv preprint arXiv:1810.01566 .

Kingma DP, Ba J (2014) Adam: A Method for Sthocastic Optimization. Proceedings of the 3rd International Conference on Learning Representations (ICLR)

Kramer MA (1991): Nonlinear principal component analysis using autoassociative neural networks. AIChE Journal. 37 (2): 233–243

Moll M and Kavraki LE (2006) Path planning for deformable linear objects. IEEE Trans. on Robotics 22(4): 625–636.

Puterman ML (2014) Markov Decision Processes.: Discrete Stochastic Dynamic Programming. John Wiley & Sons.

Schenck C, Tompson J, Fox D and Levine S (2017) Learning robotic manipulation of granular media. In: Conference on Robot Learning (CoRL).

Smolen J and Patriciu A (2009) Deformation planning for robotic soft tissue manipulation. In: Advances in ComputerHuman Interactions, 2009. ACHI'09. Second International Conferences on. IEEE, pp. 199–204.

Wada T, Hirai S, Kawamura S and Kamiji N (2001) Robust manipulation of deformable objects by a simple PID feedback. In: IEEE Int. Conf. on Robotics and Automation, ICRA.

Zhu J, Navarro B, Passama R, Fraisse P, Crosnier A and Cherubini A (2018) Robotic manipulation planning for shaping deformable linear objects with environmental contacts. IEEE Robotics and Automation Letters

Intel RealSense D400 Series Product Family Datasheet: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf

Google Colab FAQ: https://research.google.com/colaboratory/faq.html

Shadow Dexterous Hand Technical Specification: https://www.shadowrobot.com/wp-content/uploads/shadow_dexterous_hand_technical_specification_E_20190221.pdf

Rkcl framework website: http://rkcl.lirmm.net/rkcl-framework/