EMD Net: An Encode–Manipulate–Decode Network for Cloth Manipulation

Daisuke Tanaka, Solvi Arnold, and Kimitoshi Yamazaki

***Abstract*—In this letter, we present a motion planning method for automatic operation of cloth products. The problem setting we adopt here is that the current shape state of a cloth product and the shape state of the goal are given. It is necessary to decide where and what kind of manipulation is to be applied, and it is not always possible to arrive at the goal state by a single operation; that is, multiple operations might be necessary. To this problem, we propose a novel motion planning method. Our method directly connects cloth manipulations with shape changes of the cloth, by means of a deep neural network. The effectiveness of the proposed method was confirmed by simulation and real robot experiments.**

***Index Terms*—Cloth manipulation, manipulation planning, convolutional auto encoder, neural network.**

I. Introduction

In order for people to live their everyday lives, cloth products are extremely familiar. For instance, clothing and bedclothes are cloth products used everyday. Therefore, maintenance such as washing and storage is frequently necessary, and people repeat such non-productive work every day. In response to this fact, studies and developments of recognition and manipulation of cloth products have been advanced in robotics. Tasks that have been targeted include classification, unfolding, folding, etc. It can be said that automation of cloth manipulation is definitely proceeding with these results.

Nonetheless, cloth is extremely flexible and it is difficult to know the condition of the fabric itself properly. Therefore, compared with the manipulation of rigid objects, narrow task setting is necessary: for instance, the position to grasp, and the object shape transitions, are prescribed in advance. In almost all cases, the goal state of clothing shape is previously given manually. With these settings, the feasibility of manipulating flexible objects can be improved. On the other hand, as long as the predefined state can not be actually obtained, the work for obtaining that state is repeated, which is a restriction on efficiency of cloth manipulation.

The purpose of this study is to establish a manipulation planning method for folding cloth products. The problem setting we adopt here is that the current shape state of a cloth product and the shape state of the goal are given. Fig. 1 shows a conceptual diagram. In order to shift a cloth from a certain shape to a desired shape state, it is necessary to decide where and what kind of operation is to be applied. It is not always possible to arrive at the goal state by a single operation; that is, multiple operations might be necessary. When considering solving this task by automatic machines, there are two important challenges to consider. One is to enable shape prediction of fabric products. Because cloth products are deformable objects, they can take various shapes unlike a rigid body. The other is to be able to generate "manipulation" to reshape a cloth product. In order to transfer the cloth product to the desired shape state, information related to manipulations, e.g. gripping points and operation trajectories, have to be determined. Of course, these challenges are closely related.

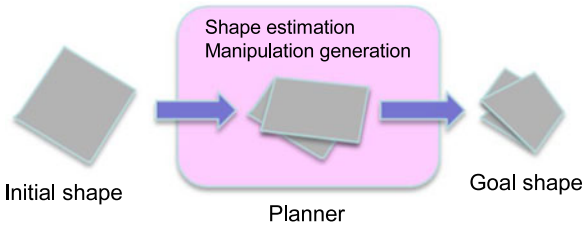

Fig. 1. A concept of folding planner.

In this letter, we focus on the folding of a rectangular cloth by an autonomous robot, and propose methods for achieving operation under the above problem setting. In our methods, even if the initial shape and the goal shape of a cloth product are set freely, appropriate operations can be planned and executed. Conventionally, in the studies of automation of cloth manipulation, there were only a few previous methods that can produce freely set goal shape states. Especially, as far as we know, there is no appropriate method for doing so online. Therefore, we propose a novel planning mechanism with real-time property by tightly coupling changes in cloth shape with the manipulations producing these, in a neural architecture.

The contributions of this letter are as follows.

- We proposed a novel method, which directly connects cloth operations with shape changes of the cloth, using a deep neural network architecture.
- We sought ways to obtain appropriate performance with a realistic amount of burden for training the above-mentioned network. A successful example is shown in this letter.

Manuscript received September 10, 2017; accepted December 26, 2017. Date of publication January 31, 2018; date of current version March 12, 2018. This letter was recommended for publication by Associate Editor L. Birglen and Editor H. Ding upon evaluation of the reviewers' comments. This work was supported by NEDO and JSPS KAKENHI under Grant 26700024. *(Corresponding author: Kimitoshi Yamazaki.)*

The authors are with the Mechanical Systems Engineering, Shinshu University, Wakasato 380-8553, Japan (e-mail: 17w4037e@shinshu-u.ac.jp; s_arnold@shinshu-u.ac.jp; kyamazaki@shinshu-u.ac.jp).

Digital Object Identifier 10.1109/LRA.2018.2800122

- We collected a part of training data through experiments using an actual robot. At that time, a data collection system was constructed so as to facilitate this work.
- We executed the task of folding a rectangular cloth into a desired shape under the restrictions of the range of motion of the actual robot.

The structure of this letter is as follows. Section II presents related work. Section III explains our manipulation planner for cloth products. In Section IV, we discuss the issues and approaches to integration of the proposed planner with real robots. Section V introduces our experimental results, Section VI discusses about the results, and Section VII lays out our conclusion and future work.

II. Related work

A. Task Oriented Cloth Manipulation

In manipulating cloth products by automatic machines, how to provide knowledge of cloth was an important issue. Since it is difficult to define a general model, previous studies have used knowledge representations tailored to the intended work. In the traditional approach, each of knowledge representations relating to global information and local information were given manually, and researchers properly used them for recognition and manipulation. Ono *et al.* [1] targeted square cloth and proposed a description of the bending state based on its contour and corners. Yamazaki [2] proposed a method of grasping points selection. Using "hem elements" and their succession, the local shape and the overall shape were described. Yuba *et al.* [3] proposed a method for unfolding a rectangular cloth placed on a table in an arbitrary unarranged shape. Cloth shape was described by the position of corners and the state of cloth wrinkles.

Osawa *et al.* [4] achieved classification of clothing type using manipulation. They observed the contour and its bottom part against hanging cloth product. The approach of hanging a cloth product by grasping one end of it is effective as a way to obtain routine information from deformable objects and is therefore frequently used in recent studies [5] [6]. In some studies, these approaches were developed into methods of selecting an appropriate grasping position from a cloth product in a suspended state. Willimon *et al.* [7] proposed a method for clothing classification. A graph-based segmentation algorithm was used in their isolation phase for categorizing the target clothing. Doumanoglou *et al.* [8] succeeded in recognizing the type and shape of clothing items during the unfolding process, using a 3D range camera. Their framework also provided the next grasping point for subsequent manipulation.

As a more general knowledge representation, methods using a three-dimensional shape model has been used. Cuén-Rochín *et al.* [9] proposed an action selection method for handling a planar cloth. The recognition method is based on matching between a 3D point cloud and a physical model, and the result is used to spread a square cloth. Kita *et al.* [10] used a 3D deformable model, and obtained a correspondence between the model and an input pointcloud that was captured by a trinocular stereo camera. Stria *et al.* [11] unfolded a clothing item by means of shape estimation using a polygonal model. Li *et al.* [12] verified a trajectory of folding by physics simulation and demonstrated their folding technique using an actual robot.

B. Coupled Knowledge Representation of Manipulation and Cloth Shape State

Above-mentioned studies, researchers considered what kind of shape state of cloth can be, and they used a model representation that is less related to manipulation. On the other hand, learning-based approaches relating to deformable object manipulation have considered a deep relationship between clothing shape and manipulation. The position of this study is close to this policy.

Tanaka *et al.* [13] proposed a method of learning motor skills. Based on topological relationship between the configuration of a dual-armed robot and deformable objects, a dressing task (putting a t-shirt on a test subject) was achieved. Lee *et al.* [14] presented force-based demonstrations of deformable object manipulation. The proposed method makes it possible to learn appropriate feedback gains to trade off position and force goals in a manner consistent with their data. Yang *et al.* [15] achieved task folding a towel. Images predicting situations slightly ahead in time were generated by a deep convolutional autoencoder, and another fully connected neural network learned the dynamics of a robot task process. These learning-based approaches have the advantage of being able to reduce the effort of the developers in designing the method. In addition, since the learning process implicitly includes generating a cloth model associated with the manipulation, it is highly suitable for online application. On the other hand, it is difficult to learn multiple work procedures at once.

III. Cloth manipulation planning

A. Planner Structure and Mechanism

The problem of manipulation planning can be formulated as follows: Given a state domain consisting of possible state set S, a manipulation (action) domain consisting of possible manipulation set M, and a state s_a and a state $s_b \in S$, find a manipulation sequence (action sequence) $p_{ab} = <m_0, \cdots, m_{n-1}>$ with $m_i \in M$ such that applying manipulation sequence p_{ab} starting in state s_a will produce state s_b.

We approach this problem using a modular neural network architecture consisting of a state-encoder module, a manipulation module, and a state-decoder module. We call this an EMD net (Encode-Manipulate-Decode) for short. A concept image of the EMD network is given in Fig. 2. The architecture is essentially a 3D convolutional auto-encoder (providing the encoder and decoder modules), with a fully connected network (the manipulation module) inserted at the bottleneck layer.

Network settings for the encoder and decoder module are as follows: 6 Layers, map counts: 1 (input), 32, 32, 64, 128, 256, 512 (output). Order reversed in decoder. Kernel size: $3 \times 3 \times 3$ (all layers). Strides: $2 \times 2 \times 1$ on the first layer (last layer in decoder), $2 \times 2 \times 2$ on all other layers. The settings of the manipulation module are as follows: 10 Layers. Input layer size:

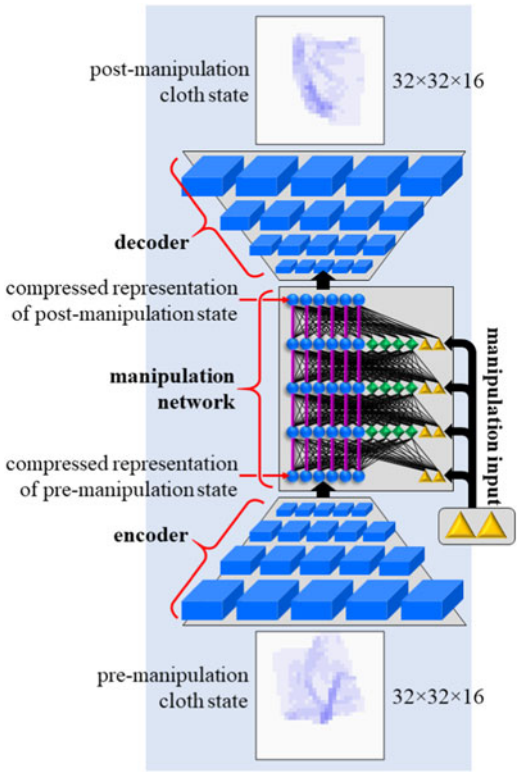

Fig. 2. Network composition during training, corresponding to $D(M(E(s_0), m_0))$ in our notation. (Concept image; actual network has more maps and neurons, see text.)

512 + 6 neurons. Hidden layer size: 512 + 512 + 6 neurons. Output layer size: 512 neurons.

Cloth state input is given in the form of a $32 \times 32 \times 16$ binary voxel representation. Each voxel takes a value of 1 if one or more vertices of the cloth mesh fall in that voxel. Voxels that are occluded from a top-down view (e.g. voxels below a 1-voxel) are also set to 1 (this is done to mimic the occlusion conditions that arise in real world data). Voxels that are neither occluded nor contain a cloth vertex take a value of 0. To ensure that the cloth is always fully in view, we apply periodic boundary conditions on the x and y axes. Before rasterisation, we multiply the vertices' z coordinates by a factor 4 to emphasize height variations. This helps to preserve creases (which do not have much height but do provide important shape information) in the voxel representation.

Manipulation input consists of 6 values: a set of 2D coordinates (x, y) for each pick-up point, and the displacement vector. All values lie in the $[-1,1]$ range. Coordinates $(-1, -1)$ and (1, 1) correspond to the upper-left and bottom-right corner of the field of view, respectively. The z-axis is not included in the manipulation representation; the height at which to pick the cloth up follows from the (x, y) coordinates, and the height to which to lift the cloth is a fixed system parameter. The dataset also includes single-handed grasps. In these examples, one of the pick-up points is undefined. When training on such manipulations, we set the coordinates for the undefined point to an arbitrary point outside the cloth.

As can be inferred from the strides and map counts given above, the encoder maps a $32 \times 32 \times 16 \times 1$ input to a $1 \times 1 \times 1 \times 512$ output, i.e. the compressed representation has no inherent spatial structure. We let it compress $32 \times 32 \times 16$ voxel representations into 512-dimensional vectors. The manipulation net takes this representation and a manipulation as input, and computes a 512-dimensional vector representing the state that results from applying the manipulation to the cloth state. The decoder then decodes this representation into a $32 \times 32 \times 16$ voxel representation of that cloth state. Below we denote application of the network modules as follows:

- $E(s_i) \rightarrow c_i$: Apply the encoder on state s_i to obtain its encoding c_i.
- $M(c_i, m_i) \rightarrow c_{i+1}$: Apply the manipulation network on encoded state c_i and manipulation m_i to obtain encoded state c_{i+1}.
- $D(c_i) \rightarrow s_i$: Apply the decoder on encoded state c_i to obtain (decoded) state s_i.

Both the autoencoder (encoder and decoder) and the manipulation network have some uncommon features. The convolution operations use periodic padding on the x and y dimension to account for the periodic boundary conditions on the voxel space. Instead of the usual zero values, we pad with the content of the opposite edges and corners of the map. Periodic padding is throughout the encoder and decoder. All connections in the encoder and decoder are initialized with random values form the $[-0.05, 0.05]$ range.

The manipulation module is comprised of three vertical sections, distinguished by neuron color and shape in Fig. 2. The blue section (round neurons) of the input layer receives the encoded state representation. In each subsequent layer, the blue section receives a copy of the activation vector on the blue section of the preceding layer (i.e. the accentuated vertical connections in Fig. 2 have fixed weights of 1.0). Activations computed in the layer are added to the copied values. The blue section essentially serves to hold the state representation as it is incrementally modified through the layers. The yellow section (triangular neurons) receives the manipulation input. Manipulation inputs are provided (identically) at every layer. This ensures that the manipulation input is available in unmodified form at every layer. The green section (diamond-shaped neurons) consists of regular neurons, without pre-specified function. It constitutes the computational resource for computing and applying the appropriate state modifications. All connection weights are initialized to random values from the $[-0.001, 0.001]$ range, except for connections from manipulation inputs, which are set to random values from the $[-0.05, 0.05]$ range instead, in order to counteract the large size difference between state and manipulation inputs (512 versus 6 values).

B. Training

We first train this net on our simulation-generated training dataset of 6000 examples, for 1,500,000 batches of 16 manipulations each. On an Nvidia GTX1080 GPU, training took approximately 3 days. Then we augment the training dataset with 400 examples of robot-generated data and train for another

400,000 batches (approx. 1 day). Note that training is on batches of individual manipulations, not on manipulation sequences. The network proved difficult to train with most commonly used update rules, but quite well trainable with the Manhattan update rule. We use two loss functions, which we denote as $loss_s$ and $loss_c$. $Loss_s$ is the mean squared error between network output, i.e. $D(M(E(s_0), m_0))$ and the voxel representation of the actual outcome, i.e. s_1.

$$loss_s = \frac{\Sigma\left[(D(M(E(s_0), m_0)) - s_1)^2\right]}{n_s}. \quad (1)$$

Where n_s is the size (number of voxels) of the state representation (16384). The network output is a real-valued voxel representation, which we compare against the actual outcome (a binary voxel representation). $Loss_c$ is introduced to force consistency of state encoding format between the input and output layers of the manipulation network.

$$loss_c = \frac{\Sigma\left[(D(M(E(s_0), m_0)) - E(s_1))^2\right]}{n_c}. \quad (2)$$

Where n_c is the size of a compressed state representation (512 with our settings). The need for consistency of state encoding format will become apparent in the next section.

We compute the gradients for both losses separately, and then combine the gradients on a perweight basis by simply summing their signs. With combining gradients, the Manhattan update rule takes the following form:

$$\delta w_i = \eta * [sign(g_i^s) + sign(g_i^c)] \quad (3)$$

Where δw_i is the change in weight for connection i, η is the learning rate, set to $1 \cdot 10^{-5}$ here, and g_i^s and g_i^c are the gradients for connection i w.r.t. $loss_s$ and $loss_c$, respectively. If the two gradients agree in sign, we update by η in the direction of that sign, otherwise the weight remains the same.

Data augmentation is performed on the fly by applying random rotation, mirroring, and grasp point swapping. The order of the grasp points is immaterial, so the outcome state remains the same. The data is further augmented with failure-to-grasp examples generated by replacing the grasp points with random points falling outside the cloth, and replacing the result state with a copy of the initial state in a randomly selected example. This teaches the net the non-effect of grasping outside the cloth. Filling the undefined pick-up points in single-handed manipulation examples with random points outside the cloth can also be considered a type of data augmentation.

C. Planning Algorithm

Next, we repurpose the trained EMD net to generate multistep manipulation sequences (plans). To compute a prediction of the outcome of applying a given n-step manipulation sequence to a given state, we can rewire the net to apply the manipulation network recurrently, n times, consuming one manipulation input at every application. For example, applying the manipulation module 3 times functionally gives us $s_3 = D(M(M(M(E(s_0), m_0), m_1), m_2))$. Fig. 3 illustrates this concept. This is repeated application of the manipulation module

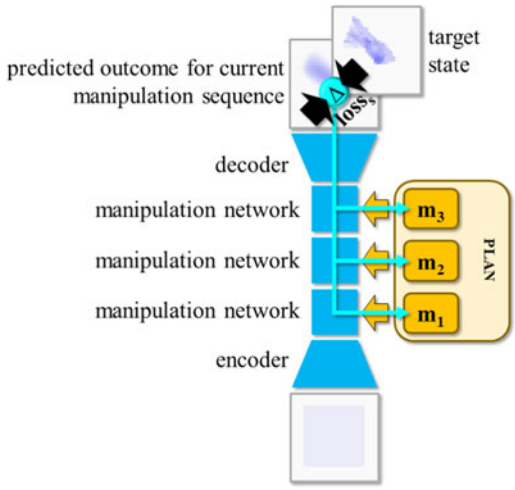

Fig. 3. Recurrent application of the manipulation network applying multistep manipulation sequences.

that requires the consistency of state encoding that we enforced during training by means of $loss_c$. Search for an n-step plan m_{ab} for transforming state s_a into state s_b is performed as follows:

1) Set the number of manipulation module applications to n.
2) Generate a random initialization for m_{ab}.
3) Feed s_a and m_{ab} into the net and perform forward propagation to obtain the predicted outcome s_p.
4) Compute $loss_s$ for s_p w.r.t. s_b.
5) Perform back propagation to obtain the gradients for the manipulation input values w.r.t. $loss_s$, and adjust m_{ab} so as to reduce $loss_s$.
6) Repeat steps 3 through 5 until the loss value stabilizes (no improvement for 25 iterations) or a set number of iterations (100 here) has been performed.

Manipulation input values are adjusted by means of the iRprop- variant [18] of the Rprop update scheme [19]. We found values of 2.0 for η^+ and 0.33 for η^- to perform well in our system. Ten instances of this search process are run in parallel on GPU (Ge-Force GTX 1080), each instance starting from a different random initialization. The search process generally takes between 2 and 9 seconds. We adopt the plan with the lowest remaining $loss_s$ value as the final result.

IV. Training Data Collection

To generate the manipulation planner, it is necessary to collect training data that records the cloth shape before and after manipulation, as well as the grasping position and the displacement thereof. Since it is difficult to collect a large amount of training data only by actual experiments, we added the dataset generated by physical simulation with actual data.

A. Data Generation by Simulation

We generate training examples using the cloth simulation functionality of the Blender 3D editor [20]. Each example consists of three elements: the pre-manipulation cloth state, the manipulation, and the post-manipulation cloth state. Manipulations

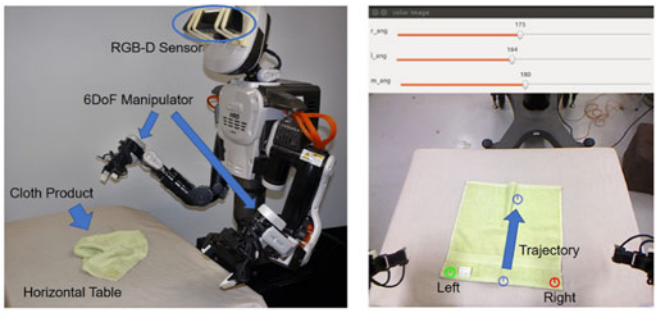

Fig. 4. A dual-armed robot with a RGB-D sensor and GUI for instructing manipulation.

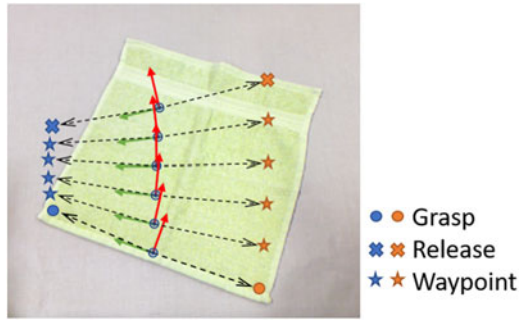

Fig. 5. An example of complemented manipulation trajectory.

consist of one or two sets of 2D coordinates indicating the location of the grasp point(s) on the XY plane and a 2D vector indicating the displacement on the XY plane.

The cloth is represented by an 80×80 mesh with the cloth modifier enabled. The neural network's field of view spans from $[-1, -1, 0]$ to $[1, 1, 0.25]$, with the cloth in its initial configuration spanning from $[-0.7, -0.7, 0.03]$ to $[0.7, 0.7, 0.03]$. The mesh itself has no explicit thickness, but we set up collision detection to keep some minimal distance between vertices, as well as between vertices and the desk surface (a plane at $z = 0$), so that the cloth behaves as if it has some thickness.

For practical reasons the robot itself is not simulated. Instead we pin cloth vertices lying in the vicinity of the grasp points to an invisible actuator object (an "empty" in Blender terminology), and assign the relevant movement trajectory to this actuator. We generate examples of single-handed and bimanual manipulation. No distinction is made between left and right hand grasps. Random grasp points are found by randomly selecting cloth vertices, and values for the displacement vector are randomly picked from the $[-1, 1]$ range.

We generate 2500 random manipulation sequences of length three as for a total of 7500 manipulation examples. Each sequence starts with the cloth laid out fully extended with the edges of the cloth parallel to the x and y axes of the coordinate system. 2000 sequences (6000 examples) are used as training data, and the remaining 500 sequences (1500 examples) as test data.

B. Data Collection Using an Actual Robot

We also collected training data by a dual-armed robot with 6-freedom manipulators, shown in Fig. 4. A horizontal table was set on front of the robot, and a cloth product was arranged in various shapes on the table. Cloth shape information was obtained as 3D pointcloud by a RGB-D camera (Microsoft KINECT) mounted on the head of the robot. A process for making the point cloud is as follows. First, the area of the cloth is extracted by means of Graphcut [21] using a color image. Then, 3D coordinates of pixels belonging to the area are calculated from 2D image coordinates and a depth value in a depth image. Finally the result is converted to voxel representation as input data introduced in Section III-A.

Manipulations of the cloth were done with either the right arm only, the left arm only, or both arms. To determine the posture of the end-effector, it is necessary to specify position variables (x, y, z) and orientation variables (ϕ, θ, ψ). However, it is a heavy burden to manually designate all these variables. To alleviate it, we reduced the degree of freedom of motion of robot so that instructions from humans can be briefly performed. The constraints were as follows:

1) The position to be gripped is limited to the silhouette edge part of the cloth.
2) The heights of the end-effector are fixed when gripping and when moving while grasping.
3) The motion of the two end-effectors is determined by the movement trajectory of one oriented point.
4) Posture relationship between the oriented point and the end-effectors is maintained while manipulation.

Based on the item 1), gripping is performed after inserting a finger under the cloth. The inserting direction is from outside of the cloth silhouette. When the cloths overlap at the gripping location, all overlapping pieces are gripped. In order to stably perform this inserting and gripping, we attached nail-like parts to robot fingers.

To determine the gripping motion, three variables, (x_m, y_m) and ψ_m, are required. The first two variables indicate the horizontal coordinates of the gripping position and the last variable means from which horizontal direction the grip is to be made with respect to the gripping position. Meanwhile, to determine the manipulation according to the conditions 3) and 4), the trajectory of (x_m, y_m) and angular variation of ψ_m from gripping to release are required. We created a GUI to concisely specify these values. A color image obtained from KINECT sensor is presented on the GUI, and an instructor specifies the position to be gripped and the position to stop gripping on this image. Then, the depth information at the specified position is referred to, and a motion planning of the manipulators is performed. At present, it does not correspond to movement with different roles for each arm, such as folding with the other arm while holding down a part of the cloth with one hand.

After instructing the gripping point and the release point using the GUI, several via-postures are set as shown in Fig. 5. These are calculated as linear interpolation of translation and rotation between grasping pose and release pose. The height of the end-effector is raised to a predetermined height from the gripping

TABLE I
RESULTS (SIMULATION DATA)

	n = 1			n = 2		
	mean	(st.dev.)	median	mean	(st.dev.)	median
Test	.026	(.014)	.023	.028	(.008)	.027
Train	.024	(.012)	.022	.031	(.012)	.028

posture to the first via-posture, and then moves horizontally in the subsequent via-postures. Then, when moving to the position to stop gripping, it is lowered the height and released the cloth after that.

In the system we implemented for motion planning, it was the default to use RRT (Rapidly-exploring Random Trees) [23]. However, RRT was unsuitable for this purpose because the reproducibility of the robot motion is not guaranteed. For this reason, we took an approach to finely arranging via-postures. By doing this, it becomes almost the same trajectory as linear interpolation.

V. EXPERIMENTS

We perform two evaluation experiments. The first experiment evaluates our system on simulation-generated test cases, with the manipulations provided by the planner being executed in Blender. The second experiment evaluates our system on robot-generated test cases, performing the manipulations provided by the planner using the HIRO robot [24], on actual cloth.

A. Simulation Experiment

We ran the planning system for 200 manipulation sequences, 100 from the training set and 100 from the test set. For each sequence $<m_1, m_2, m_3>$, we run all of its sub-sequences of length one (three subsequences: $<m_1>$, $<m_2>$ and $<m_3>$) and length two (two subsequences: $<m_1, m_2>$ and $<m_2, m_3>$). The procedure is as follows:

1) Set n to the length of the sequence, set the input state to the first state of the sequence, and set the output state to the final state of the sequence.
2) Set the manipulation network's number of loop iterations to n and run the planning process to obtain a n-step manipulation sequence.
3) Perform (in simulation) the first manipulation in the obtained manipulation sequence on the input cloth state.
4) Update the input state to the manipulation result, and reduce n by one.
5) If $n > 0$, return to step 2.

Re-running the planning process between manipulations ensures that small errors do not build up over multiple manipulations, and affords some degree of recovery when outcomes are not as expected. Table 1 shows for each set the means, standard deviations and medians of the MSE scores (i.e. mean squared difference between corresponding voxel values in the voxel representations of the outcome and goal state) we obtained for each sequence length. Medians are included because incidental catastrophic failures affect the mean. We can observe that the system generalizes well to the test data (the difference appears to be below the noise level of this measurement). Fig. 6 shows example results for 2-step manipulation sequences. Numbers above planned states and final outcome indicate loss w.r.t. the goal state. The generated manipulation sequences can be seen to produce convincing approximations of the goal states.

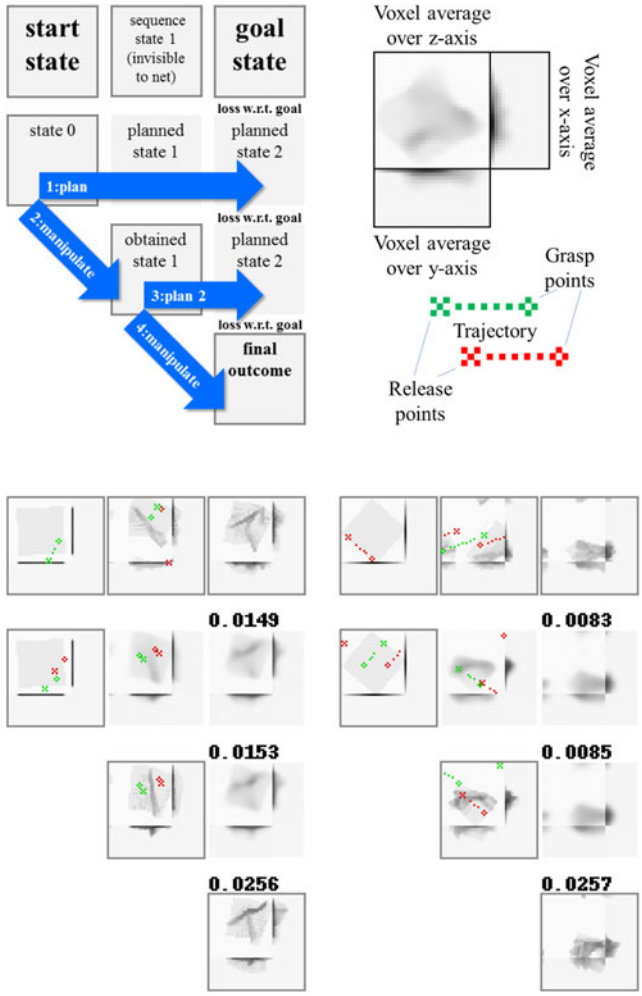

Fig. 6. Example results on 2-step sequences from the test set (simulation data).

B. Robotic Experiment

We generated 517 manipulation examples using the robotic setup, grouped in manipulation sequences of various lengths. We held out the last 100 manipulations as test data. We train the network for an additional 400,000 iterations on a mixed dataset, consisting of the 6000 items of simulation-generated training data and the 417 items of robot-generated data. We then tested the system on 10 one-step sequences and 12 two-step sequences from the robot-generated test data. The procedure is similar to that followed in the simulation experiment, but instead of performing the manipulation in simulation we performed it using the HIRO robot, on actual cloth.

For these experiments, we add an additional loss term to the back-propagation search to constrain the grasp and release

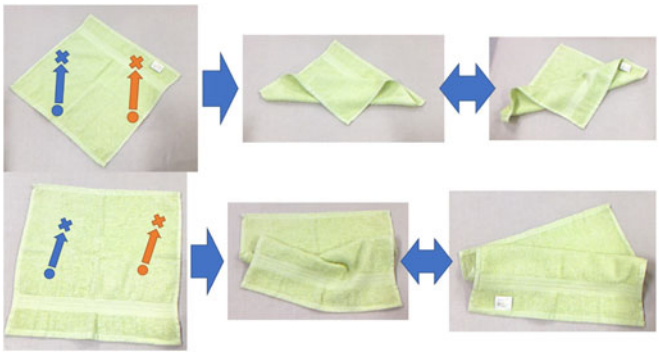

Fig. 7. Example results for manipulation on real cloth. From left to right: Initial state, result state, goal state. The two examples show reasonable approximation of the goal state.

points to an area that roughly matches the robot's range of motion. This loss term takes value 0 for grasp and release points within the range of motion, and otherwise quantifies the distance to the range of motion. This smoothly pushes points back into the range of motion when they venture outside of it during the manipulation search process.

The cloth is manually placed in a configuration closely matching the initial state of the test sequence. After each step, we obtain a point-cloud of the resulting cloth state in the same way as during data generation, and convert it to a voxel representation. The planning system generates grasp point coordinates and displacement vectors only. Deciding how to grasp the cloth at the provided coordinates is, at present, done manually. We plot the manipulation plan (grasp points and release points) over the point-cloud obtained from the robot's sensor. We then find the hem nearest to each grasp point, and set the grasp angle to be orthogonal to this hem. If grasp requires an impossible posture, we grasp from the second nearest hem instead.

Fig. 7 shows example results. The average MSE score for the robotic experiment was 0.025 (standard deviation 0.007) for one-step sequences and 0.028 (standard deviation 0.007) for two-step sequences. Whereas this score quantitatively resembles the scores for simulation data, the cloth states in the robot-generated data have a low profile compared to simulation data, causing scores to skew low in comparison. The first manipulation step of the failure case shown here reveals a common problem: The planning system's prediction of the outcome state had the cloth folded over, but actually performing the manipulation resulted in the cloth failing to bend, merely sliding some distance over the desk surface. This sliding behavior does not occur in our simulation data, which likely explains the system's failure to predict it. Failures like these stresses the need to narrow the gap between simulation-generated and robot-generated data further. As for the first two example cases, we observe that the resulting silhouettes resemble the goal states, but there are differences in internal shape. In our top-down obtained voxel representations, the height differences distinguishing these cloth configurations often proved too subtle. Obtaining more detailed height information and further boosting the height dimension in the voxel representation may improve shape agreement.

VI. Discussion

The results on simulation data suggest that the plan generation has potential. Significant strengths of our approach are that goals can be set freely, plan generation is fast (compared to explicit simulation-based planning), and we can train the network on a static database. (This in contrast to Reinforcement Learning, which requires action execution as part of the training procedure. This is time-consuming both in simulation and on robot hardware). Also, the possibility of adding novel loss terms post-training to restrict manipulation plan search in various ways is a notable strength in the context of real-world application. Whereas we limited our discussion here to sequences of length two, we find that on simulation data, sequences of length three are within reach as well: We obtained a median MSE of 0.029 on 100 test sequences of length three.

Performance on actual cloth did not match the performance on simulated cloth. We suspect that the main cause of this gap in performance lies in the substantial difference in behavior between the simulated and real cloth. Like most cloth simulations, Blender's is designed for animation purposes, e.g. waving flags and naturally moving clothes on characters, not for realistic simulation of cloth manipulation. The deformation characteristics of the real cloth proved hard to approximate in Blender. In particularly, natural cloth stiffness (i.e. resistance to stretch in the direction of the weave), and cloth-on-cloth friction proved hard to approach (e.g. folded states rarely stabilize entirely). Increasing stiffness quickly makes the simulation slow and unstable (explicitly integrated stiff cloth has a well-known tendency to explode). Stiffer cloth could be simulated stably using implicit integration techniques [22], but the reality gap will likely remain an obstacle when using mixed data. However, the system is not inherently dependent on the availability of accurate simulation data. If sufficient real-world data can be generated, simulation data becomes unnecessary. Hence we think that further automation of the robotic data generation process is the more promising avenue for improvement. The size of our simulation dataset was intentionally kept small enough that a similar set could realistically be generated on robotic hardware.

Another issue may potentially affecting performance is self-occlusion. Occlusion does not necessarily lead to ambiguity (a cloth folded in two is half occluded, but not ambiguous), but when it does, predicting the outcome of a manipulation can become somewhat of a guessing game. Simulation results with artificially imposed occlusion (as used in training here) did not show obvious deterioration compared to simulation results without occlusion. However, when training with occlusion, the generated predictions also include occlusion. Given an ambiguous goal state, the system plans to generate something that looks similar, insofar it can distinguish. This is not ideal. As noted above, on real cloth we often observed the system producing outcomes resembling the goal in silhouette but differing from it in their internal shapes. Occlusion likely plays a role here. Also, as we aim to extend plan length in future work, the system will have to deal with more complex cloth shapes with more potential for ambiguity, so some means of ambiguity reduction should be considered. A possible approach would be

to gather additional views of the cloth using a hand-mounted camera.

Other avenues of improvement we intend to pursue are inclusion of grasp angles along with the grasp coordinates in the system's manipulation output, and extension of the manipulation repertoire with separate movement trajectories for both hands.

The approach proposed here distinguishes itself from existing work in a number of ways. Unlike most work on robotic cloth manipulation, we do not constrain the task to a pre-specified manipulation sequence or outcome: any cloth configuration that can be produced with the manipulation repertoire is a potential goal state. This problem setting requires that the system is capable of representing and predicting cloth states. Explicit models (cloth simulators) provide representational and predictive ability (albeit with the caveats posed by the reality gap, as discussed above). However, free-form, multi-step planning as discussed here must consider a large number of potential manipulation sequences in order to find a suitable plan. With the settings used in this letter, plan generation considers up to 1000 sequences. Performing large numbers of manipulation sequences in explicit simulation quickly becomes too computationally expensive for real-time use (generating our simulation dataset of 2500 sequences took a few days). Hence it is perhaps not surprising that this task has, as far as we are aware, not been tackled using explicit modelling.

One way of looking at the system presented here is that the network functions as an implicit simulation that is differentiable, and therefore searchable, w.r.t. the manipulation repertoire it is trained on. This concept should be applicable to a broad variety of fuzzy planning problems.

VII. Conclusions

In this letter, we presented a motion planning method for automatic operation of cloth products. Under the problem setting in which the initial shape and the goal shape of a cloth product are given, we proposed a method to plan the gripping points and manipulation procedure. The effectiveness of the proposed method was shown by means of simulation experiments and experiments using an actual robot.

References

[1] E. Ono, H. Okabe, H. Ichijo, N. Aisaka, and H. Akami, "Robot hand with sensor for handling cloth," in *Proc. IEEE Int. Workshop Intell. Robots Syst.*, 1990, vol. 2, pp. 995–1000.

[2] K. Yamazaki, "Grasping point selection on an item of crumpled clothing based on relational shape description," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2014, pp. 3123–3128.

[3] H. Yuba, S. Arnold, and K. Yamazaki, "Unfolding of a rectangular cloth from unarranged starting shapes by a Dual-Armed robot with a mechanism for managing recognition error and uncertainty," *Adv. Robot.*, vol. 31, no. 10, pp. 554–556, 2017.

[4] F. Osawa, H. Seki, and Y. Kamiya, "Unfolding of massive laundry and classification types by dual manipulator," *J. Adv. Comput. Intell. Intell. Inf.*, vol. 11, no. 5, pp. 457–463, 2007.

[5] J. Maitin-Sphepard *et al.*, "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding," in *Proc. Int. Conf. Robot. Autom.*, 2010, pp. 2308–2315.

[6] Y. Kita, F. Saito, and N. Kita, "A deformable model driven method for handling clothes," in *Proc. Int. Conf. Pattern Recognit.*, vol. 4, 2004, pp. 3889–3895.

[7] B. Willimon, S. Birchfield, and I. Walker, "Model for unfolding laundry using interactive perception," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2011, pp. 4871–4876.

[8] A. Doumanoglou, A. Kargakos, T. Kim, S. Malassiotis, "Autonomous active recognition and unfolding of clothes using random decision forests and probabilistic planning," in *Proc. Int. Conf. Robot. Autom.*, 2014, pp. 987–993.

[9] S. Cuen-Rochin, J. Andrade-Cetto, and C. Torras, "Action selection for robotic manipulation of deformable planar objects," in *Proc. Front. Sci. Conf. Series Young Res.: Exp. Cognitive Robot.*, 2008, pp. 1–6.

[10] Y. Kita, F. Kanehiro, T. Ueshiba, and N. Kita, "Clothes handling based on recognition by strategic observation," in *Proc. Int. Conf. Humanoid Robots*, 2011, pp. 53–58.

[11] J. Stria, D. Prusa, V. Hlavac, L. Wagner, and V. Petric, "Garment perception and its folding using a dual-arm robot," in *Proc. Int. Conf. Intell. Robots Syst.*, 2014, pp. 61–67.

[12] Y. Li, Y. Yue, D. Xu, E. Grinspun, and P. K. Allen, "Folding deformable objects using predictive simulation and trajectory optimization," in *Proc. IEEE/RSJ Int., Conf. Intell. Robots Syst.*, 2005, pp. 6000 –6006.

[13] T. Matsubara, D. Shinohara, and M. Kidode, "Reinforcement learning of a motor skill for wearing a T-shirt using topology coordinates," *Adv. Robot.*, vol. 27, no. 7, pp. 513–524, 2013.

[14] A. Lee, H. Lu, A. Gupta, S. Levine, and P. Abbeel, "Learning force-based manipulation of deformable objects from multiple demonstrations," in *Proc. IEEE Conf. Robot. Autom.*, 2015, pp. 177–184.

[15] P. Yang, K. Sasaki, K. Suzuki, K. Kase, S. Sugano, and T. Ogata, "Repeatable folding task by humanoid robot worker using deep learning," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 397–403, Apr. 2017.

[16] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *Int. J Robot. Res.*, vol. 34, no. 4–5, pp. 705–724, 2015.

[17] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems." [Online]. Available: tensorflow.org. Last accessed on: Feb. 20, 2018.

[18] C. Igel and M. Hŭsken, "Improving the Rprop learning algorithm," in *Proc. 2nd Int. Symp. Neural Comput.*, 2000, pp. 115–121.

[19] M. Riedmiller and H. Braun, "RPROP – A fast adaptive learning algorithm," in *Proc. Int. Symp. Comput. Inf. Sci. VII*, 1992.

[20] [Online]. Available: https://www.blender.org/. Last accessed on: Feb. 20, 2018.

[21] Y. Boykov and V. Kolmogorov, "Computing geodesics and minimal surfaces via graph cuts," in *Proc. IEEE ICCV*, vol. 1, 2003, pp. 26–33.

[22] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Proc. 25th Annu. Conf. Comput. Graph. Interactive Tech.*, 1998, pp. 43–54, doi: http://dx.doi.org/10.1145/280814.280821

[23] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Comput. Sci. Dept., Iowa State Univ., Ames, IA, USA, Tech. Rep. TR 98–11, 1998.

[24] [Online]. Available: http://nextage.kawada.jp/en/hiro/. Last accessed on: Feb. 20, 2018.