Cloth Manipulation Planning by Back-propagation using a 3D Convolutional Auto-Encoder and a Recurrent Neural Network

* Solvi Arnold and Kimitoshi Yamazaki (Shinshu University)

1. Introduction

Within the area of manipulation planning, deformable objects pose a particularly tough challenge. Accurately predicting the result of a single manipulation can be a difficult task in itself, and even if an accurate deformation model can be obtained, predicting manipulation outcomes by means of simulation tends to be computationally expensive. This is turn makes explicit search for manipulation sequences slow and impractical. Hand-designed plans can be highly effective for specific tasks [1], but lack flexibility. Approaches retrieving and modifying manipulations from databases can produce impressive results, but involve costly matching and deformation procedures [2].

We propose a novel connectionist approach for efficient manipulation planning, based on implicit modelling of an object's deformation dynamics. Core of the system is a hybrid neural network architecture, composed of a 3D convolutional autoencoder part and a fully connected recurrent part. Given a start and goal state, the network is used to search for the manipulation sequence that produces the latter from the former, by means of error back-propagation w.r.t. the manipulation input. By setting the number of propagation loops through the recurrent section of the network, we can search for manipulation sequences of various lengths. In the present paper, we apply this manipulation planning approach to free-form manipulation of a (simulated) square cloth.

2. Task design

We adopt a free-form cloth manipulation task. The goal here is not to achieve a singular predefined outcome state, but rather to learn how a given manipulation repertoire moves the object through its state space (i.e. learn how one's actions change the object's shape and location). We use the trained net to plan manipulation sequences for realising arbitrary goal states.

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) $m_{ab} = <m_1, \dots, m_n>$ with $m_i \in$ M such that applying manipulation sequence m_{ab} starting in state s_a will produce state s_b.

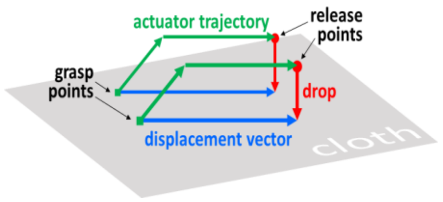

Figure 1. Schematic of cloth manipulation.

In this paper we consider as a concrete task the problem of manipulating a square piece of cloth from one configuration into another. We let states represent the cloth in some possible stable configuration. Manipulations are defined as triplets of real-valued 2D vectors. The first two vectors indicate the coordinates where the cloth is picked up (*grasp points* below), and the third vector (*displacement vector* below) indicates how much these points are moved (both points are moved by the same distance in the same direction, so one vector suffices). Figure 1 illustrates how such manipulations are translated into actuator trajectories. Plans are sequences of manipulations. The height to which the grasp points are lifted is a system parameter. One of the two grasp points may fall outside the cloth. This produces a manipulation with a single grasp point.

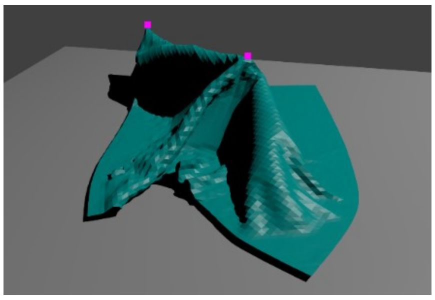

Figure 2. Snapshot from a cloth manipulation in progress in Blender. The pink squares show the points where the cloth is pinned to the (invisible) actuator.

We generate states using the cloth simulation functionality of the Blender 3D editor [3]. We lay out a square cloth on a flat surface (representing e.g. a table). We generate 2500 random manipulation sequences of length 3 as training data for a total of 7500 manipulation examples. A snapshot of a manipulation in progress is shown in Figure 2.

3. Network architecture

Our basic neural network architecture is shown in Figure 3. The architecture consists of a 3D convolutional auto-encoder, with a fully connected network sandwiched between its encoder and decoder part. We refer to the fully connected network as the manipulation network. The network is implemented in TensorFlow [4]. Network settings are as follows:

3D convolutional encoder / deconvolutional decoder:

- 6 Layers
- Map counts: 32,32,64,128,256,512 (order reversed in decoder)
- Kernel size: 3×3×3 (all layers)
- Strides: 2×2×2 (all layers)
- No weight sharing between encoder and decoder.

Manipulation network:

- 10 Layers
- Input layer size: 512+6 neurons
- Hidden layer size: 512+512+6 neurons
- Output layer size: 512 neurons
- Residual connections (pink in Figure 3) between state-encoding neurons (blue in Figure 3).

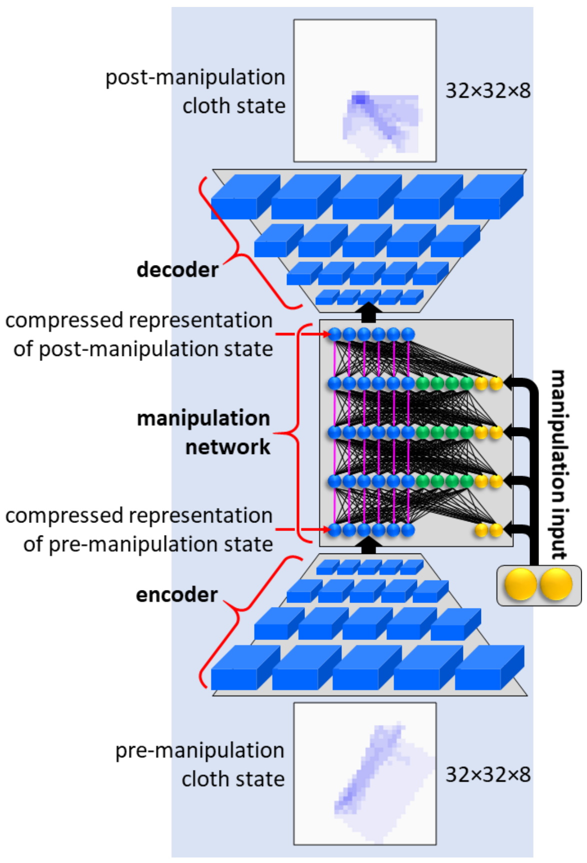
post-manipulation cloth state
32×32×8
decoder
compressed representation of post-manipulation state
manipulation network
compressed representation of pre-manipulation state
manipulation input
encoder
pre-manipulation cloth state
32×32×8

Figure 3. Network architecture (when training).

Cloth state input is given in the form of a binary voxel rasterisation of the cloth mesh obtained from Blender. The encoder compresses this 32×32×16 voxel representation into a 512-dimensional vector. 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 32×32×16 voxel representation of that cloth state. We train this net on our Blender-generated dataset. Note that training is on *individual* manipulations (not on manipulation sequences). The net is trained on one million batches of 20 manipulations each. Data augmentation is performed on the fly by applying random rotation, mirroring, and grasp point swapping. The loss function consists of the mean squared error between network output and the (voxel representation of) the actual outcome and a term promoting consistency of state encoding format between the input and output layers of the manipulation network (details of this term are omitted here for brevity). Weights are updated using the Manhattan update rule. Once trained, the network can accurately predict the result of applying a given manipulation to a given cloth state.

4. planning algorithm

Next, we use this network to generate multi-step manipulation sequences (plans). In order to generate an n-step plan, we apply the manipulation network recurrently, for n times. Figure 4 illustrates the concept for $n = 3$. In the present work, 3 is the maximum number of steps considered. Search for a plan m_{ab} for transforming state s_a into state s_b is performed as follows:

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

Manipulation input values are adjusted by means of the iRprop- update scheme [5]. 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 ran in parallel on GPU (GeForce GTX 1080), each instance starting from a different random initialisation. The search process generally takes between 2 and 9 seconds. We adopt the plan with the lowest remaining loss value as the final result.

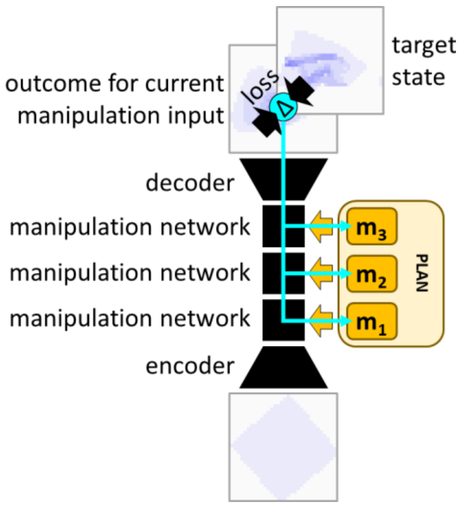

Figure 4. Using error back-propagation through recurrent application of the manipulation network to perform multi-step manipulation planning. The cyan arrows represent the back-propagation signal.

5. Performance assessment

To assess the system's performance, we generate 100 unseen test sequences, again of length 3. From a manipulation sequence of length 3 we can source 3 sequences of length 1, 2 sequences of length 2, and 1 sequence of length 3. We run the system for each such sub-sequence, as follows:

1. Set n to the length of the sequence, set the input state to the first state of the sequence as, 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 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 correction when outcomes are not as expected. Faster but less accurate performance can be achieved by planning just once and performing the obtained sequence 'blindly' (i.e. without observing and re-planning w.r.t. the intermediate results). We have not yet performed a comparison between these approaches.

	Test data [100 sequences]	Training data [50 sequences]
n = 1	.0194 (.0116)	.0194 (.0125)
n = 2	.0238 (.0102)	.0249 (.00935)
n = 3	.0254 (.00897)	.0278 (.0107)

Table 1. Average mean squared errors for manipulation outcomes w.r.t goal states for manipulation sequences of length 1, 2 and 3. Values in brackets are standard deviations.

Table 1 shows the scores obtained for the test sequences, as well as for a sampling of 50 sequences from the training data. Scores represent the mean squared errors between the goal state and the actually obtained state, both in voxel representation. The small size of the difference between the scores for training and test data indicate that our net did not overfit on the training data and generalises well to unseen data (note that the performance measured for the test data is slightly better than that for the training data, indicating that the true performance difference is below noise level for this measurement). Figure 5 shows an example of the alternating planning and execution procedure described above, applied on test data. The generated manipulation sequence can be seen to produce a convincing approximation of the goal state.

6. Future work

A limitation of the current system is that we need to set the length of the sequence to plan, which in practice will generally be unknown. Along with the manipulation plan, the system outputs a voxel representation of the expected outcome of that plan, as well as its expected loss w.r.t. the target state, providing a natural quality assessment of the plan. This can be used to automatically adjust the sequence length setting, but implementation of such a procedure remains as future work. Furthermore, we are currently in the process of integrating this planning system with a dual-armed robot [6].

Acknowledgments

This work received funding from JSPS KAKENHI (grant nr.: 26700024) and NEDO.

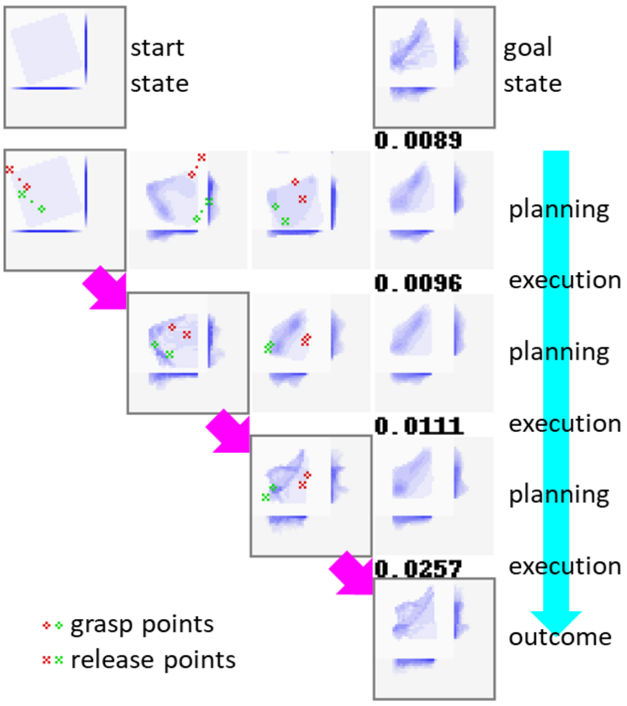

Figure 5. Representative example of interleaved planning and execution process on test data. Each image shows a top-down view and two side-views of a (translucent) voxel representation of a cloth state. Each row marked 'planning' shows the result of running the planning process for reaching the goal state from the present state. Pink arrows indicate execution (in simulation) of the first manipulation of the generated plan, updating the present cloth state. The ○ and × marks indicate grasp points and release points, respectively. Framed states are actual states, non-framed states are predicted ('imagined') states. Numbers above states indicate mean square error w.r.t. the goal state.

References

[1] Hiroyuki Yuba, Solvi Arnold and Kimitoshi Yamazaki: "Unfolding of a rectangular cloth from unarranged starting shapes by a Dual-Armed robot with a mechanism for managing recognition error and uncertainty," Advanced Robotics, Vol. 31(10): 544-556, Feb. 2017.

[2] Alex X. Lee, Abhishek Gupta, Henry Lu, Sergey Levine and Pieter Abbeel: "Learning from Multiple Demonstrations using Trajectory-Aware Non-Rigid Registration with Applications to Deformable Object Manipulation," Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5265-5272, 2015.

[3] "Blender - a 3D modelling and rendering package," www.blender.com.

[4] Martin Abadi, Ashish Agarwal, Paul Barham et al.: "TensorFlow: Large-scale machine learning on heterogeneous systems," tensorflow.org, 2015.

[5] Christian Igel and Michael Husken: "Improving the Rprop Learning Algorithm," Proceedings of the Second International Symposium on Neural Computation (NC 2000), pp.115-121, 2000.

[6] 田中大輔, Solvi Arnold, 松原崇充， 山崎公俊: "布製品折り畳みの学習と実ロボットによる操作実現," 第 35 回日本ロボット学会学術講演会， 2017.