The Surprising Effectiveness of Linear Models for Visual Foresight in Object Pile Manipulation

H.J. Terry Suh[1] and Russ Tedrake[1] **

Massachusetts Institute of Technology, Cambridge, MA 02139, USA,
hjsuh@mit.edu, russt@mit.edu

Abstract. In this paper, we tackle the problem of pushing piles of small objects into a desired target set using visual feedback. Unlike conventional single-object manipulation pipelines, which estimate the state of the system parametrized by pose, the underlying physical state of this system is difficult to observe from images. Thus, we take the approach of reasoning directly in the space of images, and acquire the dynamics of visual measurements in order to synthesize a visual-feedback policy. We present a simple controller using an image-space Lyapunov function, and evaluate the closed-loop performance using three different class of models for image prediction: deep-learning-based models for image-to-image translation, an object-centric model obtained from treating each pixel as a particle, and a switched-linear system where an action-dependent linear map is used. Through results in simulation and experiment, we show that for this task, a linear model works surprisingly well – achieving better prediction error, downstream task performance, and generalization to new environments than the deep models we trained on the same amount of data. We believe these results provide an interesting example in the spectrum of models that are most useful for vision-based feedback in manipulation, considering both the quality of visual prediction, as well as compatibility with rigorous methods for control design and analysis. Project site: https://sites.google.com/view/linear-visual-foresight/home

Keywords: Manipulation, Piles of Objects, Deformable Objects, Image Prediction, Visual Foresight, Vision-based Control

1 Introduction

The ability to predict the future is paramount to design and verification of a control policy, as modeling the dynamics of a system can enable planning and control approaches that solve tasks using the same model, or can facilitate analyzing the behavior of the closed-loop system. Conventionally in robotics, such dynamics are obtained using the laws of physics, and the state of the system is often defined as generalized coordinates of Lagrangian dynamics [18].

However, it is not always clear how to estimate the physical state, or the dynamics of how those states evolve, directly from sensor measurements. A planar

** This work was supported by NSF Award No. EFMA-1830901

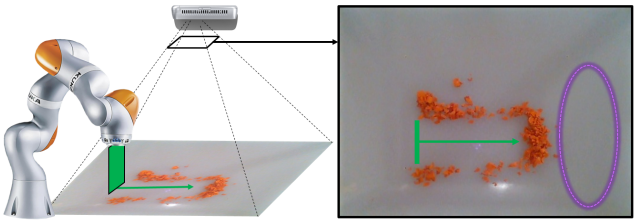

Fig. 1. Description of the task, where the robot must push all the carrots to fit in the purple target set. This real image illustrates the actual complexity of the phenomena, as well as the difficulty of observing states of each piece of carrot from the image.

pushing task [18] is a great canonical example: not only is the center of mass of the object hard to observe from vision without the assumption of uniform density [27], but the dynamics of the object is also unknown due to the uncertain interaction between the object and the table [16]. On the other hand, directly identifying the dynamics of measurements (output dynamics) offer a very general approach to vision-based control, as it relieves the need of defining true states of a system, or carefully designing an observer from visual input [8].

In this work, we deal with a task which epitomizes the strength of designing policies directly in the space of measurements: a robot observes a cutting board with diced carrots, and must find a sequence of push actions to collect them into a desired target set. The underlying physical state of the system, and even the cardinality of the state-space, is very difficult to observe from vision (Fig. 1). Despite the difficulty of modeling the problem in state space, humans trivially move around piles with ease in cooking, where diced onions and carrots are moved to make room for other ingredients. We hypothesize that this is a case of output feedback, where dynamic identification and control happens directly in the space of visual measurements.

While direct output feedback offers many advantages, the dynamics of measurements must often be obtained in a data-driven manner, as it bypasses observation of physical states. Combined with the success of deep learning in vision [14], recent works in vision-based control have heavily utilized deep-learning approaches. In [8], the term "Visual Foresight" is first used to describe output dynamics of vision, and an end-to-end neural network architecture for image-to-image translation is presented. Other works [21,7] also treat the output dynamics problem as an instance of image-to-image translation, and set up a deep-learning architecture to predict future frames.

Due to the difficulty of identifying output dynamics in high-dimensional pixel space, many of the recent works in vision-based control and intuitive physics have argued that instead of identifying the output dynamics, a neural network-based observer can observe the object-centric state of the system [9,10,26,23]. Since object-centric states are lower dimensional and have more identifiable continuous dynamics in their tasks, they were successful in showing that this approach leads to better performance and sample efficiency. However, we challenge the generality

of this approach since object states are not always of lower dimension or have simpler dynamics compared to visual dynamics. This is exemplified by our task of pushing piles of small objects, or manipulating deformable objects such as clothes or fluids. Similarly, other works have focused on identifying dynamics over keypoints [19,20], but it is not clear how to generalize keypoints to this problem where there are multiple small objects in a pile.

In [25], an object-centric approach using graph neural networks [23] is combined with visual feature vectors to manipulate piles of objects to a desired target set. While this approach may work if there are countable number of objects in the scene, they may fail to deal with partial observation of the scene due to occlusion, or a case where million particles of sand or water must be manipulated. In [6], a robot finds feasible plans to manipulate piles of dirt into a target region using A* search, with a learned Random Forest transition model over a grid-like representation. However, we have questioned if this is really the right model to fit output dynamics, and focused our attention to which class of models are more adequate for capturing the dynamics of measurements in pixel-space.

We had started to explore this problem with a deep-learning-based approach, where the dynamics are estimated in the latent space of visual feature vectors. While the prediction was visually plausible, we found that we could not succeed in achieving the downstream task due to small (in the sense of mean-squared), yet critical errors that do not agree with physical behavior. Then, we wondered how well a simple linear model would work. A linear model would offer principled ways of estimating dynamics, and provide better connection with rigorous approaches in linear systems theory. The promise of Koopman operators [24] and occupation measures [15] is that all dynamics become linear in high enough dimensions; perhaps the pixel coordinates are playing a similar role.

To investigate our question, we set up a simple controller using a Lyapunov function that operates directly on images, and evaluate the closed-loop behavior of three models: a switched-linear model where the linear map from image to image is a function of discrete inputs, variations of deep-learning image-prediction models inspired from [8,7,21], and an object-centric transport model that treats each pixel as a particle. Through evaluations in simulation and experiment, we find that the switched-linear model provides the best performance in image prediction, downstream task, and generalization to new environments.

2 Problem Statement

2.1 Setup and Notation

As illustrated in Fig. 1, our goal is to push all carrot pieces into a desired target set using visual feedback. We assume color thresholding or background subtraction so that we have a greyscale image. We denote the original greyscale image at time k as $\mathbf{I}_k \in \mathbb{R}^{N\times N}$, and its vectorized form as $y_k \in \mathbb{R}^{N^2}$. Finally, the input to the system is $u \in \mathbb{R}^4$, which is consisted of the start coordinate of the push $p_i \in \mathbb{R}^2$, the orientation of the push θ, and the push length l [25,1]. Here

we assume that the push surface is always perpendicular to the push direction. Our task is to learn the discrete-time dynamics of image prediction, which can be modeled by one of the two equivalent functions in (1).

$$\mathbf{I}_{k+1} = \bar{f}(\mathbf{I}_k, u) \qquad y_{k+1} = f(y_k, u). \tag{1}$$

Writing down the dynamics in this form also encodes our assumption of quasi-static dynamics, where friction dominates the true dynamics of the system. Our experiments with pushing piles of chopped carrots justify this assumption.

2.2 A Simple Lyapunov-based Controller

The original problem can be modeled by having continuous states $\mathbf{I}_k$ and continuous inputs u, where the goal is to drive $\mathbf{I}_0$ to some desired set of allowable images $\mathcal{S}_\mathbf{I}$. However, we choose to discretize the inputs by a grid in the action-space, performing direct search. This is due to the high non-convexity of planar pushing; most actions will not make contact with objects and produce zero gradients. This point is illustrated well in the billiard example of [12]. This setup also allows fairness for the prediction methods by allowing them to work over the same set of inputs. Solving optimal control problems for such systems with continuous states and discrete inputs often requires combinatorial search.

Fortunately, this problem admits a simple greedy strategy. To motivate this controller, imagine that the true state of the system is known, which is consisted of each particle's position $p_i \in \mathbb{R}^2$. Let $\mathcal{X} = \{p_i\}$ denote the set of these particles. Then, let $\mathcal{S}_d$ be the desired target set, which is a subset of $\mathbb{R}^2$. Our objective is to push all the particles inside $\mathcal{S}_d$. The distance from each particle to the target set is defined by looking for the closest point in the set,

$$d(p_i, \mathcal{S}_d) = \min_{p_j \in \mathcal{S}_d} \|p_i - p_j\|_p, \tag{2}$$

where $\|\cdot\|_p$ is the p-norm, a hyperparameter in the controller. Then, we simply average all the distances, and define this as a Lyapunov function on the set.

$$V(\mathcal{X}) = \frac{1}{|\mathcal{X}|} \sum_{p_i \in \mathcal{X}} d(p_i, \mathcal{S}_d). \tag{3}$$

This can be interpreted as the average Chamfer distance [3] between a discrete set of points and a continuous target set. As a sum of norms, this function is always strictly positive everywhere except for when $\forall i, d(p_i, \mathcal{S}_d) = 0$, which would mean that all the particles are inside the target set.

To extend this Lyapunov function to images, we initialize a pre-computed distance matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ where each element represents the distance between center of pixel and the target set. Let d denote the vectorized form of $\mathbf{D}$. Then, the Lyapunov function on images is evaluated as a weighted average of distances, where the weight is provided from pixels. We give two expressions for the Lyapunov function in (4), where $\odot$ represents the element-wise (Hadamard) product.

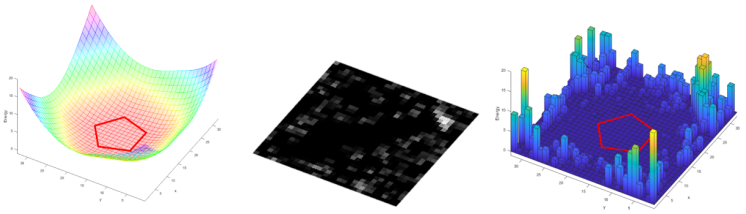

Fig. 2. Visualization of the Lyapunov function with a pentagonal target set and using the $p = 2$ distance norm. The pre-initialized $\mathbf{D}$ matrix (left) gets multiplied with the image (center), and the average value of elements is taken from the result (right).

$$\bar{V}(\mathbf{I}_k) = \frac{1}{\|\mathbf{I}_k\|_{1,1}} \mathbf{D} \odot \mathbf{I}_k \qquad V(y_k) = \frac{1}{\|y_k\|_1} d^T y_k. \tag{4}$$

$\bar{V}(\mathbf{I}_k)$ is zero if and only if indices of the non-zero pixels of the images coincide with the indices of zero elements of the $\mathbf{D}$ matrix, which would signify that non-zero pixels are only placed in the target set.

Finally, we argue that the $\bar{V}(\mathbf{I}_k)$ is a Control Lyapunov Function (CLF), since it satisfies the following property, if we assume $\bar{f}$ is known:

$$\forall \mathbf{I} \notin \mathcal{S}_\mathbf{I}, \quad \exists u \quad \text{s.t.} \quad \Delta\bar{V} = \bar{V}(\mathbf{I}_{k+1}) - \bar{V}(\mathbf{I}_k) = \bar{V}(\bar{f}(\mathbf{I}_k, u)) - \bar{V}(\mathbf{I}_k) < 0, \tag{5}$$

where $\mathcal{S}_\mathbf{I}$ is the set of goal images. This is due to the fact that for every image that is not in the target set, we can always find a small particle to push towards the target set and decrease the value of the Lyapunov function. Given this Lyapunov function in (4), we choose a greedy feedback policy that minimizes the Lyapunov function from its current value at every given timestep.

$$u^* = \arg\min_u \bar{V}(\bar{f}(\mathbf{I}_k, u)). \tag{6}$$

The intuitive explanation of this controller is to enforce the closed-loop behavior of the system to resemble a bowl with the target set as the flat bottom region (see Fig. 2). Deforming the cutting board into a bowl, the carrot pieces will naturally fall on the target set. We hypothesize that depending on the accuracy of the prediction model $\mathbf{I}_{k+1} = \bar{f}(\mathbf{I}_k, u)$, the ability to descend along the Lyapunov function will differ. A better model should be able to descend faster along the Lyapunov function, and we use this descent curve as a task-relevant benchmark of the predictive capability of different models.

Furthermore, we note that if the set $\mathcal{S}_d$ is non-convex in Cartesian coordinates of $\mathbb{R}^2$, then $V(\mathcal{X})$ becomes non-convex as well. However, in image space, $V(y_k)$ is still linear (thus globally convex) function if $\|\mathbf{I}_k\|_{1,1}$ is relatively constant for all k (4). This process is similar to convex relaxations, where originally non-convex problems are convexified in the space of distributions. Thus, a simple greedy strategy will stabilize to even non-convex target sets. Finally, we exclude sets that cannot be stabilized due to inherent mechanical limitation of the system, such as the width of the pusher.

3 Models for Image Prediction

3.1 Switched-Linear Model

Model Description. In this model, we assume a switched-linear system [5], with the following form:

$$y_{k+1} = \mathbf{A}_i y_k, \tag{7}$$

where $\mathbf{A}_i \in \mathbb{R}^{N^2 \times N^2}$ and $i = \{1, \cdots, |\mathcal{U}|\}$ with $\mathcal{U}$ the discretized action space. The action is directly represented as choosing the $\mathbf{A}$ matrix. One of the challenges of working in image space is that the coordinates of the actual object is recorded in the indices of the pixels, not the actual pixel values. Then, for a given action, such a linear map $\mathbf{A}$ can act as a permutation matrix that transports pixels in indices.

Learning of Dynamics via Least Squares. To train the model, we collect many pairs of (y_{k+1}, y_k) that are all *subject to the same action*. Let there be M such pairs. Then, we construct a data matrix by appending the column vectors horizontally. Thus, we have two matrices $\mathbf{Y}_{k+1}, \mathbf{Y}_k \in \mathbb{R}^{N^2 \times M}$, where the i^{th} column of $\mathbf{Y}_k$ is the vectorized image before the push, and the i^{th} column of $\mathbf{Y}_{k+1}$ is the corresponding vectorized image after the push. Then, optimal identification of the transition matrix $\mathbf{A}$ can be written as a solution to the least squares optimization problem:

$$\mathbf{A}^* = \underset{\mathbf{A}}{\arg\min} \quad \|\mathbf{Y}_{k+1} - \mathbf{A}\mathbf{Y}_k\|_F, \tag{8}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. This is a matrix Ordinary Least Squares (OLS) problem, and has a standard closed-form solution.

We attempt to also add constraints or add regularization terms and quantify which least-squares method performs the best. We questioned if imposing a permutation matrix-like structure could provide meaningful regularization for the $\mathbf{A}$ matrix, and improve the test performance of prediction. We also considered non-negativity constraints, or equality constraints such that the rows or columns sum up to 1. These optimization problems can be easily formulated as an instance of Quadratic Programming (QP). The general formulation of these constraints is written down in (9).

$$\begin{aligned} \mathbf{A}^* = \underset{\mathbf{A}}{\arg\min} \quad & \|\mathbf{Y}_{k+1} - \mathbf{A}\mathbf{Y}_k\|_F \\ \text{subject to} \quad & \mathbf{A} \geq 0, \quad \textstyle\sum_i \mathbf{A}[i,j] = 1 \quad \forall j. \end{aligned} \tag{9}$$

While the transcription to QP is trivial, the number of decision variables in $\mathbf{A} \in \mathbb{R}^{N^2 \times N^2}$ is N^4, which can reach a million even for a 32×32 resolution image. Thus, the full-scale problem is very time-consuming to solve. However, for row-constrained or non-negative constraints, we can utilize the following relation between the row of $\mathbf{Y}_{k+1}$ and the row of $\mathbf{A}$:

$$\mathbf{Y}_{k+1}[i,:] = \mathbf{A}[i,:]\mathbf{Y}_k, \tag{10}$$

where $\mathbf{A}[i,:]$ denotes the i^{th} row of $\mathbf{A}$. Using this relation, it is possible to decompose this optimization problem into N^2 optimization problems with N^2 decision variables, since the optimal solution of $\mathbf{A}[i,:]$ is the optimal solution for the entire problem.

While the column-sum-constrained problem also presents an interesting interpretation as a Markov stochastic matrix [11], it no longer possesses this natural decomposition, and the resulting problem is too big to handle for regular QP solvers, requiring more scalable formulations such as ADMM [22,4]. We plan to explore more methods of dealing with large-scale constrained least squares in future works.

Tensor Description of the Transition Matrix. To interpret the $\mathbf{A}$ matrix, let us reshape $\mathbf{A} \in \mathbb{R}^{N^2 \times N^2}$ into a tensor $\bar{\mathbf{A}} \in \mathbb{R}^{N \times N \times N \times N}$. Similar to how $\mathbf{A}[i,j]$ denotes how much the j^{th} element of y_k affects the i^{th} element of y_{k+1}, $\bar{\mathbf{A}}[i,j,m,n]$ signifies how much the pixel value of $\mathbf{I}_k[m,n]$ affects the pixel value of $\mathbf{I}_{k+1}[i,j]$. Using this relation, we can rewrite (7) as

$$\mathbf{I}_{k+1} = \bar{\mathbf{A}} \times_2 \mathbf{I}_k, \tag{11}$$

where $\times_2$ denotes the 2-mode tensor product over the last two dimensions.

When the matrix is reshaped as a tensor, the structure of the tensor reveals an interesting fact about the linear model. The rows of the $\mathbf{A}$ matrix can be reshaped as an $N \times N$ image, as $\mathbf{A}[i,:]$ corresponds to $\bar{\mathbf{A}}[i,j,:,:]$ in the tensor representation. Then, the matrix $\bar{\mathbf{A}}[i,j,:,:]$ gets element-wise multiplied with the original image, and the grand sum of this product matrix becomes the value of the pixel in $\mathbf{I}_k[i,j]$. We call the image $\bar{\mathbf{A}}[i,j,:,:]$ the kernel of the pixel $\mathbf{I}_k[i,j]$, and use it as a convenient visualization of the learned linear model in the results section (Fig. 8).

Affine Transformation on Input Images. The above process explains how to find the optimal $\mathbf{A}$ for a given action u, but we need to learn every $\mathbf{A}$ for all

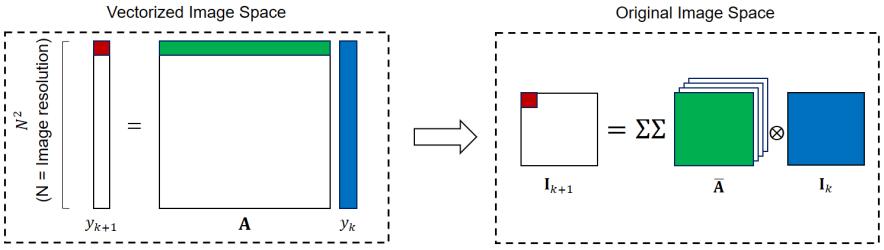

Fig. 3. In the vectorized image space, each element (red) is the result of a dot product between the corresponding row of the $\mathbf{A}$ matrix (green) and the original vector (blue). By refactoring the green row vector as an image, we can represent the red pixel as a sum of element-wise multiplication between the green image and the blue image.

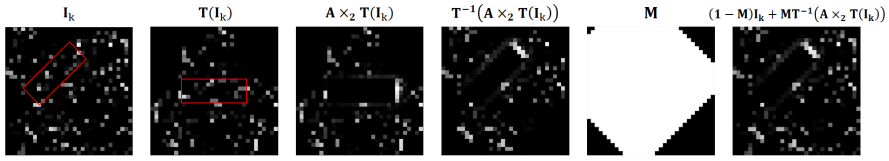

Fig. 4. Affine Transformation Process for Reducing Action Space Dimension

the different inputs. We note that our action space is in a Cartesian frame, and it should be possible to utilize coordinate transforms to reduce the dimension of the action space, similar to the approach taken in [28].

Given an action u, we can create a rectangle that represents the area swept by the pusher, which we call the "push rectangle". Then, we compute $\mathbf{T} \in SE(2)$, an affine transformation from the center-of-image frame to pusher rectangle frame. Let $\mathbf{T}(\mathbf{I})$ be the transformed image. We apply our linear map to the transformed image, and transform the predicted image back to original coordinates by the inverse transform $\mathbf{T}^{-1}$. To deal with the missing parts of images in the transformation process, we let a mask go through the same affine transformation $\mathbf{M} = \mathbf{T}^{-1}\left(\mathbf{T}\left(\mathbf{1}^{N\times N}\right)\right)$, which is used to combine the predicted image and the original image. This process is illustrated in Fig. 4.

With the affine transforms on input, we only need to compute different $\mathbf{A}_i$ matrices for the length of the push, which significantly decreases the amount of data we need to have. We discretize push length into 5, and train an $\mathbf{A}$ matrix for each one of them with 1000 sample pairs of $(\mathbf{I}_k, \mathbf{I}_{k+1})$.

3.2 Deep-Learning Model

We use a deep-learning model based on [8,7,21,9], which was our original approach to this problem. This architecture (Fig.5) computes the latent-space vectors of the image $\mathbf{I}_k$ through a convolutional autoencoder [17], approximates the dynamics on this latent-space vector with multiple layers of Multi-Layer Perceptrons (MLP), then decodes the resulting latent-space vector to obtain the predicted image $\mathbf{I}_{k+1}$. In addition, similar to [8,7,21], skip connections between matching dimensions of the convolutional autoencoder are added to preserve high-frequency features that are lost during convolution. This architecture is shown in Fig.5, and we will label it *DVF-Original*. The network is trained with $23,000$ pairs of $(\mathbf{I}_k, u, \mathbf{I}_{k+1})$.

Although majority of similar architectures append the action with the latent-space states and compute the dynamics, we wanted to make a fair comparison with the linear model which utilizes affine transforms on the input image, such that the network is also learning image-to-image translation instead of image×action-to-image. Thus, we utilize the same method illustrated in Fig.4, and use 5 separate networks for different push lengths, similar to the mixture-of-experts approach. The action-appending part of the network is removed in order to facilitate an image-to-image architecture. We call this method *DVF-Affine*.

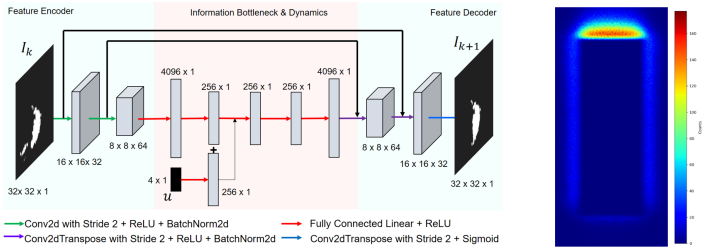

Fig. 5. Left: deep-learning Architecture for the DVF-Original Model. Right: Learned distribution of the object-centric model.

Let the forward predicting function be $\hat{\mathbf{I}}_{k+1} = \bar{f}(\mathbf{I}_k, u)$. Each network is trained end-to-end in a supervised manner using the true outcome image, and the loss function is defined as:

$$\mathcal{L}(\hat{\mathbf{I}}_{k+1}, \mathbf{I}_{k+1}) = \|\hat{\mathbf{I}}_{k+1} - \mathbf{I}_{k+1}\|_F = \|\bar{f}(\mathbf{I}_k, u) - \mathbf{I}_{k+1}\|_F. \tag{12}$$

The samples that were used for training the linear model is used to train each network in DVF-Affine. The training was done using the ADAM optimizer [13], and each network went through more than 1000 epochs. The first 500 epochs are trained with a step learning rate scheduler, starting with $\eta = 0.01$ and decreasing by a factor of 10 every 100 epochs ($\gamma = 0.1$). This process is repeated again for another 500 epochs, resuming from the result of the first 500 epochs.

3.3 Object-Centric Transport Model

Although the true states of the objects are unobservable, we attempt to build a first-principles model by treating each non-zero pixel in the image as an object. We assume that pixels that are within the swept area of the push are teleported by a probability distribution around the pusher. The algorithm works as follows:

1. From image $\mathbf{I}_k$, collect the coordinates of the non-zero pixels as a set $\mathcal{X}_k$
2. Divide $\mathcal{X}_k$ into $\mathcal{X}_a$ (affected) and $\mathcal{X}_u$ (unaffected) depending on whether or not the pixel coordinate is within the push rectangle.
3. From a bivariate distribution $P(u)$, sample $|\mathcal{X}_a|$ coordinates $p_i \sim P$, and denote the new set of sampled coordinates as $\mathcal{X}_n$.
4. Create a new set $\mathcal{X}_{k+1} = \mathcal{X}_n \cup \mathcal{X}_u$
5. Evaluate the particle Lyapunov function in (3) directly.

We approximate the distribution $P(u)$ by a small uniform distribution right in front of the push rectangle. To justify this choice, we transform each image using the affine transformation from Fig. 4, evaluate the difference between the image $(\mathbf{I}_{k+1} - \mathbf{I}_k)$, then threshold the difference above zero. This distribution is illustrated in the right side of Fig. 5, and we can see that the uniform distribution approximates this distribution well.

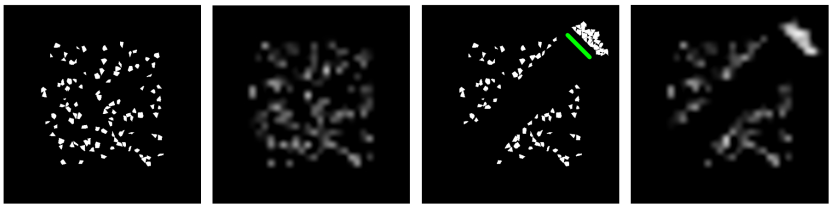

Fig. 6. The left two images visualize the simulator environment at time k, and the downsampled image $\mathbf{I}_k$ that acts as input to the prediction algorithm. The right two images corresponds to time $\mathbf{I}_{k+1}$ after the push.

4 Simulation Results

To learn prediction models and test the closed-loop performance, we built a simulator using Pymunk. Each carrot piece is randomly generated by sampling points along a fixed-size circle and computing their convex hull. The simulator is displayed in Fig. 6.

4.1 Switched-Linear Model

Least-Squares Comparison Results. We compare different methods for least-squares from Sec. 3.1. Around 1000 pairs of images $(\mathbf{I}_k, \mathbf{I}_{k+1})$ are collected, where each image is 32×32. 800 pairs are used to estimate the optimal value of $\mathbf{A}$ using the CVXOPT solver [2], while 200 pairs are used as test set. This result is shown in Fig. 7, and the non-negative least-squares formulation worked best for the estimation of the linear map. On the other hand, the row sum-constrained least-squares solution seem to regularize the transition matrix excessively, as the sum of all kernels are forced to be equal to 1. Therefore, we adopt non-negative least squares as the default estimation scheme for the $\mathbf{A}$ matrix.

Learned Kernels of the Linear Model. Following our interpretation of the $\mathbf{A}$ matrix as a tensor in Fig. 3, we visualize each of the kernels to see if the kernel images are interpretable. The result of this visualization is shown in Fig. 8

We see that for the areas outside the push rectangle, the kernel learns the identity transform. For areas inside the push rectangle, the kernel values are

Fig. 7. Test error on different least-squares algorithms

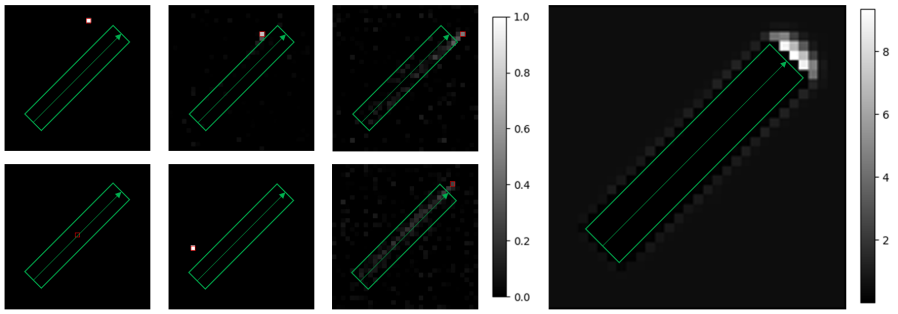

Fig. 8. Visualization of different kernels for different pixel locations. The red pixel represents the location of the pixel of the kernel, and the green rectangle represents the pushed area. The right image represents the "step response" of the matrix.

almost zero, as it learns that pixels will be gone from this location. Finally, for pixels that are at the edge of the push rectangle, it learns a kernel to weight the values inside the pushed area and sum them up to place them in front of the pushed area.

In addition, we initialize y_k to all 0.5 (half of maximum value), and see the step response of $y_{k+1} = \mathbf{A}y_k$ to see the behavior of the $\mathbf{A}$ matrix. The result is displayed in the right side of Fig. 8, and we see that the transition matrix learns the correct behavior of the action u. It is surprising to see the similarity between this step response and the probability distribution obtained in the transport model (Fig. 5), given that the linear model is obtained entirely from data while the transport model is relatively hand-written.

4.2 Comparison of Visual Prediction

We compare the results of visual prediction using the three models. For fairness, all models utilize a 32×32 resolution image. The predictions are visualized in Fig. 9. We observe that both the linear model and deep models predict a reasonable result of the push. The Object-Centric model is visualized with locations of particles as it does not synthesize future image frames.

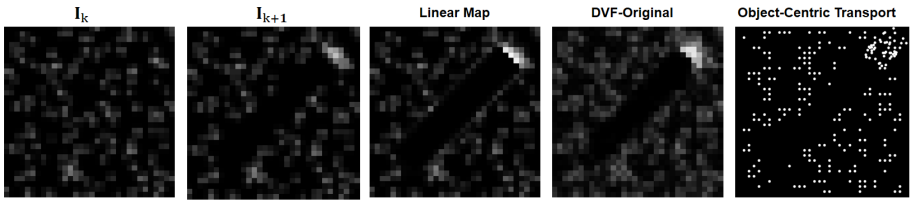

Fig. 9. Visual comparison of Model Prediction Results.

In order to quantify performance, we set up 1000 test examples generated from different initial conditions and actions. We take $\|\mathbf{I}_{k+1} - f(\mathbf{I}_k, u)\|_F$ as the error metric, and average the error across all 1000 samples. This test error is illustrated in Table. 1. Although the deep models were trained for over 1000 epochs, the linear model has the lowest prediction error. Given that the linear model is a subclass of the deep model and has less parameters compared to the deep model, the fact that it performs better is surprising.

We believe this is an empirical evidence for some underlying linearity in the problem, and thus the linear model has better inductive bias. Furthermore, choosing a linear model allowed us to train the model (9) to global optimality, and add meaningful regularization constraints such as non-negativity.

Table 1. Test Prediction Error, Model Details, and Evaluation Time

Model Name	Dimension	Parameters	Samples	Test Error
Switched-Linear	$\mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$	1,048,576	5,000	1.858
DVF-Original	$\mathbb{R}^{N\times N} \times \mathbb{R}^4 \to \mathbb{R}^{N\times N}$	2,382,721	23,000	2.062
DVF-Affine	$\mathbb{R}^{N\times N} \to \mathbb{R}^{N\times N}$	2,317,185	5,000	2.537

4.3 Closed-Loop Performance

We compare the performance of the models in closed-loop by evaluating the best action on a fixed grid of u according to (6). The goal of the task is to push all the pieces into the blue region and drive the Lyapunov value V to 0. Some image trajectories are visualized in Fig. 10, and the result is plotted in Fig. 11.

We observed that while the linear model, the object-centric transport model, and DVF-Affine model are able to converge to the target set, DVF-Original failed to do so. A common failure mode in the DVF-Original model is that its optimal predicted action does not cause any change in the actual scene, and the same

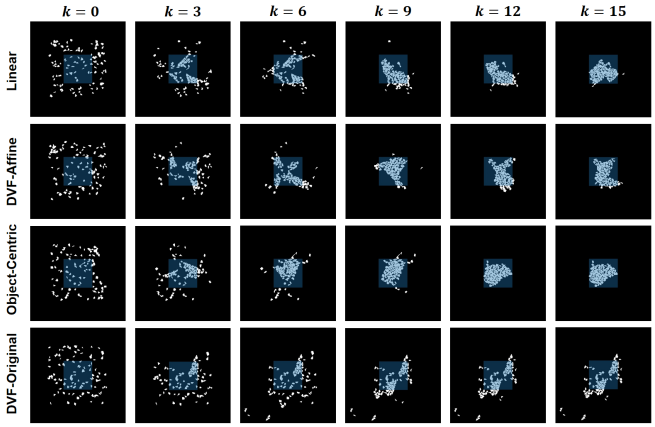

Fig. 10. Visualization of the closed-loop behavior using three predicted models. The blue square region denotes the target set.

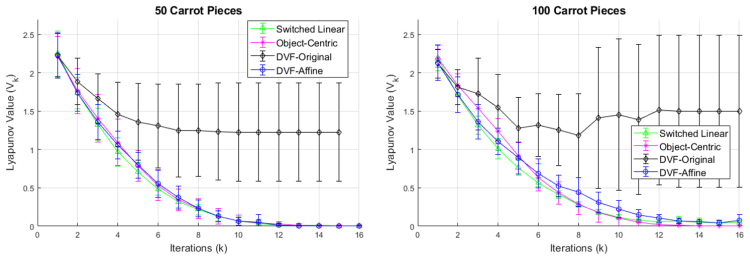

Fig. 11. Evaluation of descent along the Lyapunov function for different methods. The goal of the task is to make $V = 0$. Each method was repeated for 10 times on different initial conditions.

image gets feedbacked, making the closed-loop system get stuck in a loop. This failure mode agrees with the observation in [25], where policies trained in the latent-space of spatial autoencoders [9] repeatedly produced actions that did not change the scene. We observed these failure cases and found that deep networks mispredict some key physical behavior that is apparent in object pushing, such as making carrots disappear instead of pushing them.

There was no significant performance difference between the remaining three models with small number of carrots. However, for larger number of carrot pieces, the linear model started showing a steeper descent curve compared to the DVF-Affine and object-centric models, with DVF-Affine usually requiring 2 more iterations before convergence. On average, the switched-linear model took 1.00 second of computation, objected-oriented took 27.00s, and DVF-affine model took 4.00s on the CPU, while the DVF-original took 0.17s on GPU.

We attempt more difficult target sets to showcase the ability of the linear model, as shown in Fig. 12. Although it takes more iterations, the prediction of the linear model is successful in converging to a complex non-convex target set when coupled with the controller.

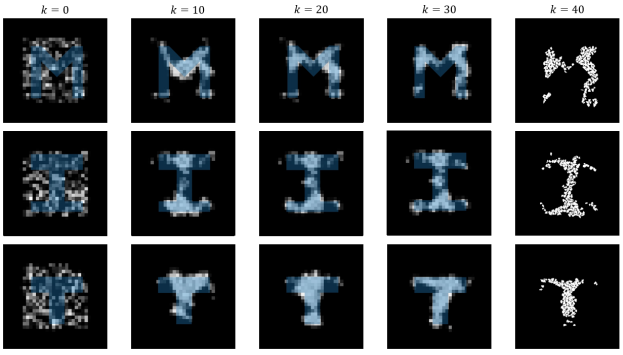

Fig. 12. Evaluation of linear model on more difficult target sets. The last images are in original simulator resolution.

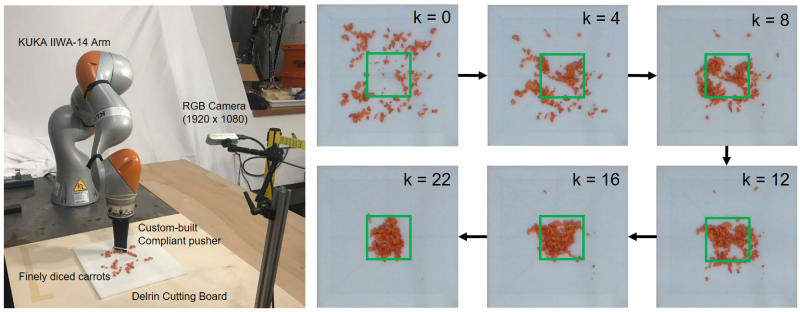

Fig. 13. Left: Experiment of the setup. Right: Visualization of the closed-loop behavior of the linear model.

5 Experiment Results

To test our algorithm in the real-world, we prepare an experiment setup illustrated in Fig. 13. The initial piles are generated by manually spraying the carrot pieces on the board, and the result is averaged over multiple runs. We use the dynamics obtained in simulation directly on the experiment without additional fine-tuning, as we believe this sim-to-real transfer process will be a good measure of the model's ability to generalize.

From the result plot of Fig. 14, we see that the linear model and the object-centric model was still able to converge to the target set. But surprisingly, the *DVF-Affine model did not succeed* in the real-world task, despite its success in simulation. This suggests that the DVF-Affine model overfitted to the training images provided by the simulator, while the switched-linear model learned a generalizable model that can extend to new environments.

What could have allowed the linear model to generalize to the real environment while the DVF-Affine model did not? Given that they are trained with the sample training samples, we believe that the linear model provided better inductive bias for the phenomenon, which empirically signifies an inherent linearity in the problem.

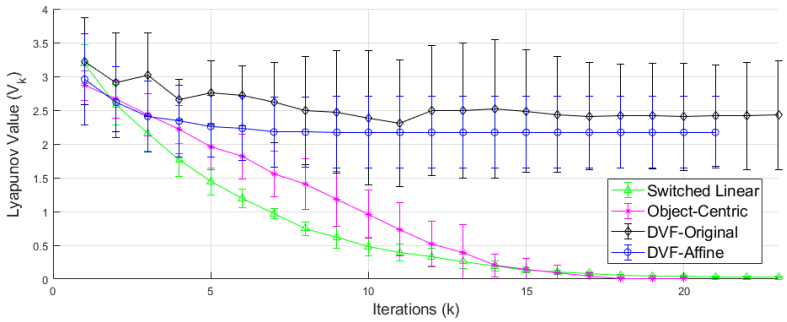

Fig. 14. Result of the experiments. Each method was repeated 5 times.

6 Conclusion

In this work, we have proposed a switched-linear model that utilizes action-dependent linear maps to predict image dynamics. We compared the performance of our model with deep-learning-based models which estimate the dynamics on the latent space of images. We found that that the linear model outperformed the deep model in test prediction error, closed-loop performance, and generalization. Furthermore, through comparing the linear model's performance with the object-centric model, we found that output feedback offers a competitive alternative to our object-centric first-principles approach.

As linear models are a subclass of deep models, we believe that given enough training samples, the right architecture, and good hyperparameters, there exists a deep model that will outperform the linear model. However, the search procedure for such models is not understood well. On the other hand, linear models allow globally optimal learning in their parameter space, and can easily be regularized through constraints. Furthermore, this relatively simple task serves to illustrate that high-dimensional visual dynamics, that may not seem linear at first glance, show promise to be approximated well by tractable linear models.

We wish to better investigate where this linearity comes from, and how general this approach will be for manipulation tasks such as pushing rigid bodies, tasks in 3D, and tasks that require utilizing the color-space. We aim to understand what model classes are most useful for vision-based feedback in manipulation, considering both the quality of visual prediction, as well as compatibility with rigorous methods for system identification, control design, and analysis.

References

1. Agrawal, P., Nair, A., Abbeel, P., Malik, J., Levine, S.: Learning to poke by poking: Experiential learning of intuitive physics. CoRR **abs/1606.07419** (2016)
2. Andersen, M., Dahl, J., Vandenberghe, L.: Cvxopt: A python package for convex optimization, version 1.1.6 (2013)
3. Barrow, H.G., Tenenbaum, J.M., Bolles, R.C., Wolf, H.C.: Parametric correspondence and chamfer matching: Two new techniques for image matching. In: Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2, Morgan Kaufmann Publishers Inc. (1977) 659–663
4. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1) (January 2011) 1–122
5. Colaneri, P.: Analysis and control of linear switched systems. Politecnico di Milano (2015)
6. Elliott, S., Cakmak, M.: Robotic cleaning through dirt rearrangement planning with learned transition models. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). (2018) 1623–1630
7. Finn, C., Goodfellow, I., Levine, S.: Unsupervised learning for physical interaction through video prediction. In: Proceedings of the 30th International Conference on Neural Information Processing Systems, Red Hook, NY, USA (2016) 64–72
8. Finn, C., Levine, S.: Deep visual foresight for planning robot motion. CoRR **abs/1610.00696** (2016)

9. Finn, C., Tan, X.Y., Duan, Y., Darrell, T., Levine, S., Abbeel, P.: Learning visual feature spaces for robotic manipulation with deep spatial autoencoders. CoRR **abs/1509.06113** (2015)
10. Fragkiadaki, K., Agrawal, P., Levine, S., Malik, J.: Learning visual predictive models of physics for playing billiards. CoRR **abs/1511.07404** (2015)
11. Gagniuc, P.: Markov Chains: From Theory to Implementation and Experimentation. Wiley (2017)
12. Hu, Y., Anderson, L., Li, T., Q, S., Carr, N., Ragan-Kelley, J., Durand, F.: Difftaichi: Differentiable programming for physical simulation. CoRR **abs/1412.6980** (2014)
13. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6) (May 2017) 84–90
15. Lasserre, J.B., Henrion, D., Prieur, C., Trélat, E.: Nonlinear Optimal Control via Occupation Measures and LMI-Relaxations. SIAM Journal on Control and Optimization **47**(4) (January 2008) 1643–1666
16. Ma, D., Rodriguez, A.: Friction variability in planar pushing data: Anisotropic friction and data-collection bias. IEEE Robotics and Automation Letters **3**(4) (Oct 2018) 3232–3239
17. Masci, J., Meier, U., Cireşan, D., Schmidhuber, J.: Stacked convolutional auto-encoders for hierarchical feature extraction. In Honkela, T., Duch, W., Girolami, M., Kaski, S., eds.: Artificial Neural Networks and Machine Learning, Berlin, Heidelberg, Springer Berlin Heidelberg (2011) 52–59
18. Mason, M.T.: Mechanics and planning of manipulator pushing operations. The International Journal of Robotics Research **5**(3) (1986) 53–71
19. Minderer, M., Sun, C., Villegas, R., Cole, F., Murphy, K., Lee, H.: Unsupervised learning of object structure and dynamics from videos. CoRR (2019)
20. Qin, Z., Fang, K., Zhu, Y., Li, F.F., Savarese, S.: Keto: Learning keypoint representations for tool manipulation. ArXiv **abs/1910.11977** (2019)
21. Sarkar, M., Pradhan, P., Ghose, D.: Planning robot motion using deep visual prediction. CoRR **abs/1906.10182** (2019)
22. Umenberger, J., Manchester, I.R.: Scalable identification of stable positive systems. In: IEEE 55th Conference on Decision and Control (CDC). (Dec 2016) 4630–4635
23. Watters, N., Tacchetti, A., Weber, T., Pascanu, R., Battaglia, P.W., Zoran, D.: Visual interaction networks. CoRR **abs/1706.01433** (2017)
24. Williams, M.O., Kevrekidis, I.G., Rowley, C.W.: A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition. Journal of Nonlinear Science **25**(6) (December 2015) 1307–1346
25. Wilson, M., Hermans, T.: Learning to manipulate object collections using grounded state representations. Conference on Robot Learning (2019)
26. Ye, Y., Gandhi, D., Gupta, A., Tulsiani, S.: Object-centric forward modeling for model predictive control. ArXiv **abs/1910.03568** (2019)
27. Yong Yu, Fukuda, K., Tsujio, S.: Estimation of mass and center of mass of graspless and shape-unknown object. In: IEEE International Conference on Robotics and Automation. Volume 4. (May 1999) 2893–2898 vol.4
28. Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A., Funkhouser, T.A.: Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. CoRR **abs/1803.09956** (2018)