Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control

Frederik Ebert*, Chelsea Finn*, Annie Xie, Sudeep Dasari, Alex Lee, Sergey Levine

Abstract—Deep reinforcement learning (RL) algorithms can learn complex robotic skills from raw sensory inputs, but have yet to achieve the kind of broad generalization and applicability demonstrated by deep learning methods in supervised domains. We present a deep RL method that is practical for real-world robotics tasks, such as robotic manipulation, and generalizes effectively to never-before-seen tasks and objects. In these settings, ground truth reward signals are typically unavailable, and we therefore propose a self-supervised model-based approach, where a predictive model learns to directly predict the future from raw sensory readings, such as camera images. At test time, we explore three distinct goal specification methods: designated pixels, where a user specifies desired object manipulation tasks by selecting particular pixels in an image and corresponding goal positions, goal images, where the desired goal state is specified with an image, and image classifiers, which define spaces of goal states. Our deep predictive models are trained using data collected autonomously and continuously by a robot interacting with hundreds of objects, without human supervision. We demonstrate that visual MPC can generalize to never-before-seen objects—both rigid and deformable—and solve a range of user-defined object manipulation tasks using the same model.

Index Terms—Deep Reinforcement Learning, Video Prediction, Robotic Manipulation, Model Predictive Control

1 Introduction

Humans are faced with a stream of high-dimensional sensory inputs and minimal external supervision, and yet, are able to learn a range of complex, generalizable skills and behaviors. While there has been significant progress in developing deep reinforcement learning algorithms that learn complex skills and scale to high-dimensional observation spaces, such as pixels [1], [2], [3], [4], learning behaviors that *generalize* to new objects, goals, or scenes remains an open problem. The key to generalization is diversity. When deployed in a narrow, closed-world environment, a reinforcement learning algorithm will recover skills that are successful only in a narrow range of settings. Learning skills in diverse environments, such as the real world, presents a number of significant challenges: external reward feedback is extremely sparse or non-existent, and the agent has only indirect access to the state of the world through its senses, which, in the case of a robot, might correspond to cameras and joint encoders. We approach the problem of learning generalizable behavior in the real world from the standpoint of sensory prediction. Prediction is often considered a fundamental component of intelligence. If we predict raw sensory observations directly, we do not need to assume availability of low-dimensional state information or an extrinsic reward signal. Through prediction, it is possible to learn useful concepts about the world even from a raw stream of sensory observations, such as images from a camera. Image observations are both information-rich and high-dimensional, presenting both an opportunity and a challenge. Future observations provide a substantial amount of supervisory information for a machine learning algorithm. However, the predictive model must have the capacity to predict these high-dimensional observations, and the control algorithm must be able to use such a model to effectively select actions to accomplish user-specified goals. Examples for the user-specified goals we consider are shown in figure 1.

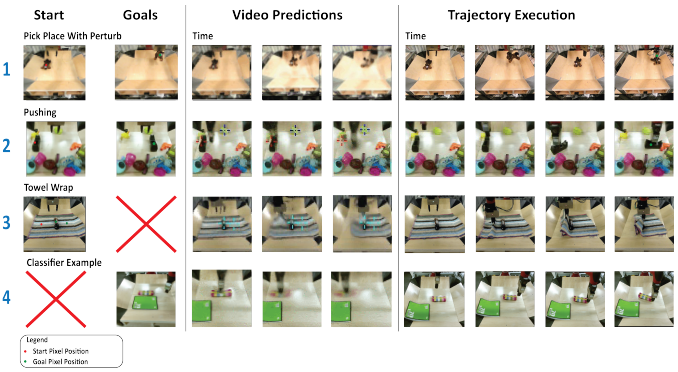


Fig. 1: Visual MPC generalizes to a wide range of tasks and objects, allowing flexibility in goal specification and both rigid and deformable objects not seen during training. Each row shows a different example trajectory. From left to right, we show the task definition, the video-predictions for the planned motion, and the actual executions. Here a task is defined by moving an object marked by a designated pixel, shown by a red dot to a goal position shown by a green dot. A goal-image with the desired goal-configuration can be provided both in combination with a designated pixel or instead of it. Best viewed in PDF.

- *The first two authors contributed equally.*

Manuscript received xx.xx.xxx

We study control via prediction in the context of robotic manipulation, formulating a model-based reinforcement learning approach centered around prediction of raw sensory observations. One of the biggest challenges in learning-based robotic manipulation is generalization: how can we learn models that are useful not just for a narrow range of tasks seen during training, but that can be used to perform new tasks with new objects that were not seen previously? Collecting a training dataset that is sufficiently rich and diverse is often challenging in highly-structured robotics experiments, which depend on human intervention for reward signals, resets, and safety. We instead set up a minimally structured robotic control domain, where data is collected by the robot via unsupervised interaction with a wide range of objects, making it practical to collect large amounts of interaction data. The robot collects a stream of raw sensory observations (image pixels), without any reward signal at training time, and without the ability to reset the environment between episodes. This setting is both realistic and necessary for studying RL in diverse real-world environments, as it enables automated and unattended collection of diverse interaction experience. Since the training setting affords no readily accessible reward signal, learning by prediction presents an appealing option: the supervision signal for prediction is always available even in the stream of unsupervised experience. We therefore propose to learn action-conditioned predictive models directly on raw pixel observations, and show that they can be used to accomplish a range of pixel-based manipulation tasks on a real robot in the physical world at test-time.

The main contributions of this work are as follows. We present *visual MPC*, a general framework for deep reinforcement learning with sensory prediction models that is suitable for learning behaviors in diverse, open-world environments (see figure 2). We describe deep network architectures that are effective for predicting pixel-level observations amid occlusions and with novel objects. Unlike low-dimensional representations of state, specifying and evaluating the reward from pixel predictions at test-time is nontrivial: we present several practical methods for specifying and evaluating progress towards the goal—including distances to goal pixel positions, registration to goal images, and image classifiers—and compare their effectiveness and use-cases. Finally, our evaluation shows how these components can be combined to enable a real robot to perform a range of object manipulation tasks from raw pixel observations. Our experiments include manipulation of previously unseen objects, handling multiple objects, pushing objects around obstructions, handling clutter, manipulating deformable objects such as cloth, recovering from large perturbations, and grasping and maneuvering objects to user-specified locations in 3D-space. These results represent a significant advance in the *generality* of skills that can be acquired by a real robot operating on raw pixel values using a single model.

This article combines and extends material from several prior conference papers [5], [6], [7], [8], presenting them in the context of a unified system. We include additional experiments, including cloth manipulation and placing tasks, a quantitative *multi-task benchmark* assessing the performance of our method on a wide range of tasks, as well as a comprehensive, open-sourced simulation environment to facilitate future research and better reproducibility. The code and videos can be found on the following webpage[1].

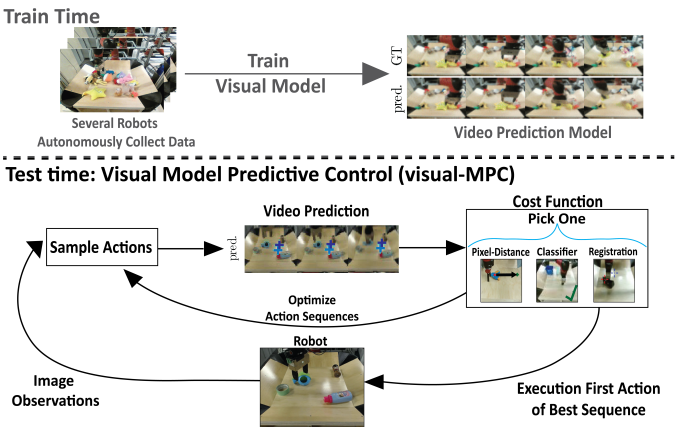


Fig. 2: Overview of visual MPC. At training time (top) interaction data is collected autonomously and used to train a video-prediction model. At test time (bottom) this model is used for samnpling-based planning. In this work we discuss three different choices for the planning objective.

2 Related Work

Learning without access to rewards. Reinforcement learning typically assumes access to an external reward signal to *reinforce* good behavior [9], [10]. Our approach learns a *goal-agnostic* model without any external rewards, and then uses this model to plan to achieve user-specified goals at test-time. A wide range of tasks can be solved by defining appropriate *internal* objectives. These internal objectives are computed based on roll-outs of the sensory prediction model, as detailed in section 5. In fact there have been several recent works on RL formulations that do not require external rewards [11], [12], or reduce the necessity for them [13].

Model-based reinforcement learning. Learning a model to predict the future, and then using this model to act, falls under the general umbrella of model-based reinforcement learning. Model-based algorithms are generally known to be more efficient than model-free methods [14], [15], and have been used with both low-dimensional [16] and high-dimensional [17] model classes. However, model-based RL methods that directly operate on raw image frames have not been studied as extensively. Several algorithms have been proposed for simple, synthetic images [18] and video game environments [19], [20], [21], but have not been evaluated on generalization or in the real world, and recent work has also studied model-based RL for individual robotic skills [22]. In contrast to these works, we place special emphasis on *generalization*, studying how predictive models can enable a real robot to manipulate previously unseen objects and solve new tasks. Several prior works have also sought to learn inverse models that map from pairs of observations to actions, which can then be used greedily to carry out short-horizon tasks [23], [24], [25]. However, such methods do not directly construct longer-term plans, relying instead on

1. For videos & code: https://sites.google.com/view/visualforesight

greedy execution. In contrast, our method learns a forward model, which can be used to plan out a sequence of actions to achieve a user-specified goal.

Self-supervised robotic learning. A number of recent works have studied self-supervised robotic learning, where large-scale unattended data collection is used to learn individual skills such as grasping [26], [27], [28], [29], [30] or obstacle avoidance [31], [32]. In contrast to these methods, our approach learns predictive models that can be used to perform a variety of manipulation skills, and does not require a success measure, event indicator, or reward function during data collection.

Sensory prediction models. We propose to leverage sensory prediction models, such as video-prediction models, to enable large-scale self-supervised learning of robotic skills. Prior work on video prediction has studied predicting synthetic video game images [21], [33] and real-world images [34], [35], [36], using both direct autoregressive frame prediction [35], [36], [37] and latent variable models [22], [38]. Video prediction without actions has been studied for unstructured videos [37], [39], [40] and driving [41], [42], as well as more structured 3D point cloud data [43]. Several works have sought to use more complex distributions for future images, for example by using autoregressive models [36], [44]. While this often produces sharp predictions, the resulting models are extremely demanding computationally. In this work, we extend video prediction methods that are based on predicting a transformation from the previous image [35], [42].

3 Overview

In this section, we summarize our visual model-predictive control (MPC) approach, which consists of a model-based reinforcement learning approach to end-to-end learning of robotic manipulation skills. Our method, outlined in figure 2, consists of three phases: unsupervised data collection, predictive model training, and planning-based control via the model at test-time.

Unsupervised data collection: At *training time*, data is collected autonomously by applying random actions sampled from a pre-specified distribution. It is important that this distribution allows the robot to visit parts of the state space that are relevant for solving the intended tasks. Therefore we design a custom sampling distribution to direct data collection to the desired parts of the state space, as further detailed in Sections 7 and 9.4.

Model training: A video-prediction model is trained on the collected data offline. The video-prediction model takes as input an image of the current time-step and a sequence of actions, and generates the corresponding sequence of future frames. The video-prediction model is detailed in Section 4.

Test-time control: At *test time* we use a sampling-based, gradient free optimization procedure, similar to "shooting methods", to find the sequence of actions that minimizes a cost function. Further details, including the motivation for this type of optimizer, can be found in Section 6.

Depending on how the goal is specified, we use one of the following three cost functions: When the goal is provided by clicking on an object and a desired goal-position, a *pixel-distance cost-function*, detailed in Section 5.1,

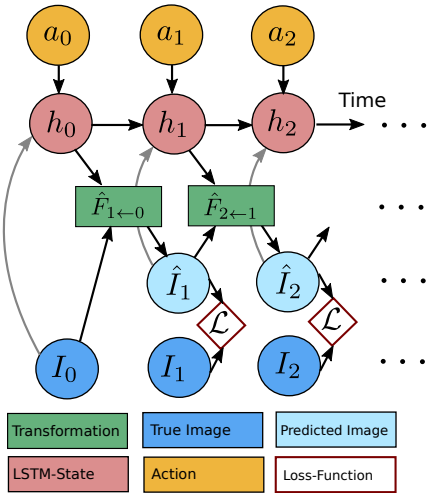


Fig. 3: Computation graph of the video-prediction model. Time goes from left to right, a_t are the actions, h_t are the hidden states in the recurrent neural network, $\hat{F}_{t+1\leftarrow t}$ is a 2D-warping field, I_t are real images, and $\hat{I}_t$ are predicted images, $\mathcal{L}$ is a pairwise training-loss.

is used that evaluates how far the designated pixel is from the goal pixels. We can specify the goal more precisely by providing a goal-image in addition to the pixel positions and make use of *image-to-image registration* to compute a cost function, this is explained in Section 5.2. Finally we show that more abstract tasks can be solved by providing one or several demonstrations and employing a classifier-based cost function as detailed in Section 5.3. The strengths and weaknesses of different costs functions and trade-offs between them are discussed in Section 5.4.

The model is used to plan T steps into the future, and the first action of the action sequence that attained lowest cost, is executed. In order to correct for mistakes made by the model, the actions are iteratively replanned at each real-world time step[2] $\tau \in \{0, ..., \tau_{max}\}$ following the framework of model-predictive control (MPC). In the following sections, we explain the video-prediction model, the planning cost function, and the trajectory optimizer.

4 Video Prediction for Control

In visual-MPC we use a transformation-based video-prediction architecture, first presented in [35]. The advantage of using transformation based-models over a model that directly generates pixels, is that prediction is easier when only few objects in the image move by relatively small amounts and also the transformations can be leveraged to obtain predictions of *where* certain pixels in the image are moving, a property that is used in several of our planning cost-function formulations. The model, which is implemented as a recurrent neural network g_θ parameterized by θ, has a hidden state h_t and takes in a previous image and an action at each step of the rollout. Future images $\hat{I}_{t+1}$ are generated by warping the previous generated image $\hat{I}_t$ or

2. With real-world step we mean timestep of the real-world as opposed to predicted timesteps.

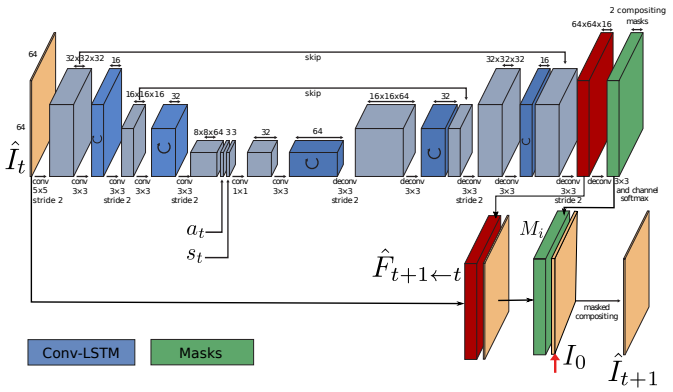


Fig. 4: Forward pass through the recurrent SNA model based on Equation (5). The red arrow indicates where the image from the first time step I_0 is concatenated with the transformed images $\hat{F}_{t+1\leftarrow t} \diamond \hat{I}_t$ multiplying each channel with a separate mask to produce the predicted frame for step $t+1$.

the previous true image I_t, when available, according to a 2-dimensional flow field $\hat{F}_{t+1\leftarrow t}$. A simplifying illustration of model's structure is given in figure 3. It is also summarized in the following two equations:

$$[h_{t+1}, \hat{F}_{t+1\leftarrow t}] = g_\theta(a_t, h_t, I_t) \quad (1)$$

$$\hat{I}_{t+1} = \hat{F}_{t+1\leftarrow t} \diamond \hat{I}_t \quad (2)$$

Here bilinear sampling operator $\diamond$ interpolates the pixel values bilinearly with respect to a location (x, y) and its four neighbouring pixels in the image, similar to [45]. Note that as shown in figure 3, at the first time-step the real image is transformed, whereas at later timesteps previously generated images are transformed. The model is trained by standard backpropagtion-through time by performing gradient descent on a mean-squared error type image reconstruction loss, denoted by $\mathcal{L}$ in figure 3.

A forward pass of the RNN is illustrated in figure 4. We use a series stacked convolutional LSTMs and standard convolutional layers interleaved with average-pooling layers. The result of this computation is the 2 dimensional flow-field $\hat{F}_{t+1\leftarrow t}$ which is used to transform a current image I_t or $\hat{I}_t$.

Predicting motion of individual pixels: When using visual-MPC with a cost-function based on start- and goal pixel positions, a model is required that can effectively predict the 2-D motion of the user-selected start pixels $d_0^{(1)}, \ldots, d_0^{(P)}$ up to T steps into the future. Since the model we employ is transformation based, this motion prediction capability emerges implicitly, and therefore no external pixel motion supervision is required. To predict the future positions of the designated pixel d, the same transformations which are used to transform the images are applied to the distribution over designated pixel locations. The warping transformation $\hat{F}_{t+1\leftarrow t}$ can be interpreted as a stochastic transition operator allowing us to make probabilistic predictions about future locations of individual pixels:

$$\hat{P}_{t+1} = \hat{F}_{t+1\leftarrow t} \diamond \hat{P}_t \quad (3)$$

Here P_t is a distribution over image locations which has the same spatial dimension as the image. For simplicity we assume that we only use a single designated pixel. At the first time step the distribution $\hat{P}_0$ is defined as 1 at the position of the user-selected designated pixel and zero elsewhere. The distribution $\hat{P}_{t+1}$ is normalized at each prediction step.

Since this basic model, which we refer to as dynamic neural advection (DNA) model, predicts images only based on the previous image, it is unable to recover shapes (e.g., objects) after they have been occluded, for example by the robot arm. Therefore, this model is only suitable for planning motions where the user-selected pixels are not occluded during the manipulation, which restricts its use in cluttered environments or with multiple selected pixels. In the next section, we introduce an enhanced type of model, which lifts this limitation by employing temporal skip connections.

Note that when using a planning cost function that does not depend on the prediction of pixel positions, like a classifier-based cost function, as detailed in section 5.3, we do not require the model to output transformations and virtually any video prediction model can be used.

Skip Connection Neural Advection Model To enable effective tracking of objects through occlusions, we can extend the model discussed in the previous section with temporal skip connections: we now transform pixels not only from the previously generated image $\hat{I}_t$, but from all but from all previous images $\hat{I}_1, \ldots \hat{I}_t$, including the context image I_0, which is a real image. All these transformed images can combined to a form the predicted image $\hat{I}_{t+1}$ by taking a weighted some over all transformed images, where the weights are given by masks $\mathbf{M}_t$ with the same size as the image and a single channel:

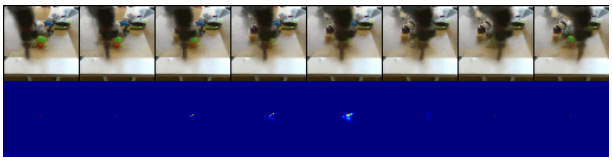
$$\hat{I}_{t+1} = \mathbf{M}_0(\hat{F}_{t+1\leftarrow 0} \diamond I_t) + \sum_{j=1}^{\tau} \mathbf{M}_j(\hat{F}_{t+1\leftarrow j} \diamond \hat{I}_j). \quad (4)$$

We refer to this model as the *skip connection neural advection model (SNA)*, since it handles occlusions by using temporal skip-connections such that when a pixel is occluded (e.g., by the robot arm or by another object) it can still reappear later in the sequence.

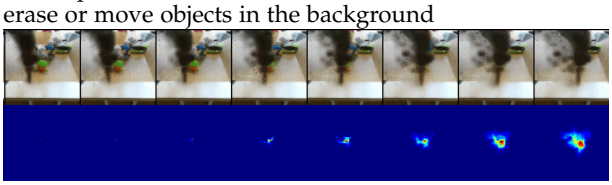
Transforming from all previous images comes with increased computational cost, since the number of masks and transformations scales with the number of time-steps τ. However, we found that in practice a greatly simplified version of this model, where transformations are applied only to the previous image and the *first image* of the sequence I_0 works equally well. Moreover we found that transforming the first image of the sequence is not necessary, as the model uses its pixels primarily to generate the image background. Therefore, we can use the first image directly, without transformation, such that

$$\hat{I}_{t+1} = \mathbf{M}_0 I_0 + \mathbf{M}_1(\hat{F}_{t+1\leftarrow t} \diamond \hat{I}_t). \quad (5)$$

Here, we make the assumption that occluded objects are static throughout the prediction horizon. This assumption allows us to dispense the intermediate transformations and only provide a skip connection from the very first image in the sequence I_0, which is also the only real image, since



(a) Skip connection neural advection (SNA) does not erase or move objects in the background



(b) Standard DNA [5] exhibits undesirable movement of the distribution $P_d(t)$ and erases the background

Fig. 6: Top rows: Predicted images of arm moving *in front of* green object with designated pixel (as indicated in Figure 5). Bottom rows: Predicted probability distributions $P_d(t)$ of designated pixel obtained by repeatedly applying transformations.

all of the subsequent images are predicted by the model. Hence, this model only needs to output 2 masks.

We provide an example of the model recovering from occlusion in Figure 6. In this figure, the arm is predicted to move in front of the designated pixel, marked in blue in Figure 5. The predictions of the DNA model, shown in figure Figure 6(b), contain incorrect motion of the marked object, as shown in the heatmaps visualizing $\hat{P}_t$, although the arm actually passes in front of it. This is because the DNA model cannot recover information about an object that it has 'overwritten' during its predictions, causing the model to predict that the pixel *moves with the arm*. We identified this as one of the major causes of planning failure using the DNA model. By contrast our SNA model predicts that the occluded object will not move, shown in figure Figure 6(a).

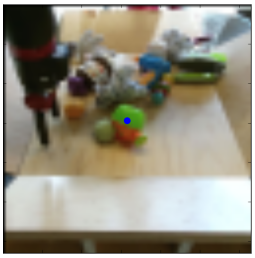


Fig. 5: The blue dot indicates the designated pixel

5 Planning Cost Functions

In this section we present three different kinds of planning cost functions, each expecting different kinds of user input.

5.1 Pixel-Distance based Cost

A convenient way to define a robot task is by choosing one or more pixels in the robot's camera view and choosing a destination where each pixel should be moved. For example, the user might select a pixel on an object and ask the robot to move it 10 cm to the left. The advantages of this type of task definition are that success can be measured quantitatively in a straight-forward way, as detailed in section 9.

Given a distribution over pixel positions P_0 at time $t = 0$, the model predicts distributions over its positions P_t at time $t \in \{1, \ldots, T\}$. One way of defining the cost per time-step c_t is by using the expected euclidean distance to the goal point d_g, which is straight-forward to calculate from P_t and g, as follows:

$$c = \sum_{t=1,\ldots,T} c_t = \sum_{t=1,\ldots,T} \mathbb{E}_{\hat{d}_t \sim P_t} \left[\|\hat{d}_t - d_g\|_2 \right] \qquad (6)$$

The per time-step costs c_t are summed together giving the overall planing objective c. The expected distance to the goal provides a smooth planning objective and enables longer-horizon tasks, since this cost function encourages movement of the designated objects into the right direction for each step of the execution, regardless of whether the goal-position can be reached within T time steps or not. This cost also makes use of the uncertainty estimates of the predictor, when computing the expected distance to the goal. For multi-objective tasks with multiple designated pixels $d^{(i)}$ the costs are summed to together, and optionally weighted according to a scheme discussed in Section 5.2.

5.2 Registration-Based Cost

When using pixel distance-based cost functions it is necessary to know the initial locations $d_0^{(1)}, \ldots, d_0^{(P)}$ at each time-step. To update the belief of where the target object currently is, we register the current image to the start and optionally also to a *goal image*, where the designated pixels are marked by the user. Adding a goal-image can make visual-MPC more precise, since when the target object is close to the goal position, registration to the goal-image greatly improves the position estimate of the designated pixel.

Crucially, the registration method we introduce is self-supervised, using the same exact data for training the video prediction model and the registration model. This allows both the predictor and registration model to continuously improve as the robot collects more data.

Before further detailing our learned registration system, we discuss a simple alternative approach for obtaining a cost-function for video-prediction based control: One naïve approach could be to use the pixel-wise error, such as MSE, between a *goal image* and the *predicted image*. However there is a severe issue with this approach: When objects in the image are far from the position in the goal image (e.g., they do not overlap) there is no gradient signal with respect to changes in the actions, therefore optimization becomes impractical.

Test Time Procedure We will first describe the registration scheme at test time (see Figure 8(a)). We separately register the current image I_t to the start image I_0 and to the goal image I_g by passing it into the registration network R, implemented as a fully-convolutional neural network. The registration network produces a flow map $\hat{F}_{0\leftarrow t} \in \mathbb{R}^{H \times W \times 2}$, a vector field with the same size as the image, that describes the relative motion for every pixel between the two frames.

$$\hat{F}_{0\leftarrow t} = R(I_t, I_0) \qquad \hat{F}_{g\leftarrow t} = R(I_t, I_g) \qquad (7)$$

The flow map $\hat{F}_{0\leftarrow t}$ can be used to warp the image of the current time step t to the start image I_0, and $\hat{F}_{g\leftarrow t}$ can be used to warp from I_t to I_g (see Figure 7 for an illustration). There is no difference to the warping operation used in the video-prediction model, explained in section 4, equation 2:

$$\hat{I}_0 = \hat{F}_{0\leftarrow t} \diamond I_t \qquad \hat{I}_g = \hat{F}_{g\leftarrow t} \diamond I_t \qquad (8)$$

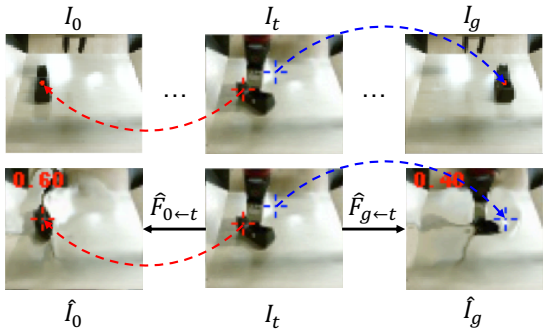
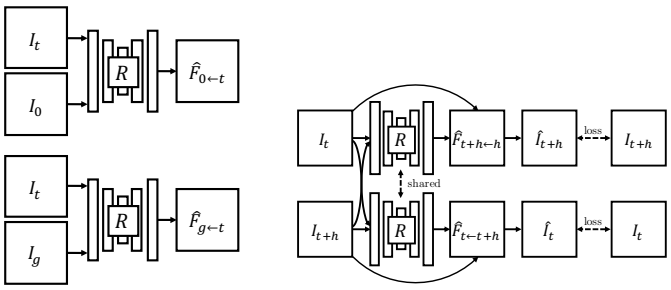


Fig. 7: Closed loop control is achieved by registering the current image I_t globally to the first frame I_0 and the goal image I_g. In this example registration to I_0 succeeds while registration to I_g fails since the object in I_g is too far away.



(a) Testing usage. (b) Training usage.

Fig. 8: (a) At test time the registration network registers the current image I_t to the start image I_0 (top) and goal image I_g (bottom), inferring the flow-fields $\hat{F}_{0\leftarrow t}$ and $\hat{F}_{g\leftarrow t}$. (b) The registration network is trained by warping images from randomly selected timesteps along a trajectory to each other.

In essence for a current image $\hat{F}_{0\leftarrow t}$ puts I_t in correspondence with I_0, and $\hat{F}_{g\leftarrow t}$ puts I_t in correspondence with I_g. The motivation for registering to both I_0 and I_g is to increase accuracy and robustness. In principle, registering to either I_0 or I_g is sufficient.

While the registration network is trained to perform a global registration between the images, we only evaluate it at the points d_0 and d_g chosen by the user. This results in a cost function that ignores distractors. The flow map produced by the registration network is used to find the pixel locations corresponding to d_0 and d_g in the current frame:

$$\hat{d}_{0,t} = d_0 + \hat{F}_{0\leftarrow t}(d_0) \qquad \hat{d}_{g,t} = d_g + \hat{F}_{g\leftarrow t}(d_g) \tag{9}$$

For simplicity, we describe the case with a single designated pixel. In practice, instead of a single flow vector $\hat{F}_{0\leftarrow t}(d_0)$ and $\hat{F}_{g\leftarrow t}(d_g)$, we consider a neighborhood of flow-vectors around d_0 and d_g and take the median in the x and y directions, making the registration more stable. Figure 9 visualizes an example tracking result while the gripper is moving an object.

Registration-Based Pixel-Distance Cost Registration can fail when distances between objects in the images are large. During a trajectory, the registration to the first image typically becomes harder, while the registration to the goal image becomes easier. We propose a mechanism that estimates which image is registered correctly, allowing us to utilize only the successful registration for evaluating the planning cost. This mechanism gives a high weight λ_i to pixel-distance costs c_i associated with a designated pixel $\hat{d}_{i,t}$ that is tracked successfully and a low, ideally zero, weight to a designated pixel where the registration is poor. We propose to use the photometric distance between the true frame and the warped frame evaluated at $d_{0,i}$ and $d_{g,i}$ as an estimate for *local* registration success. A low photometric error indicates that the registration network predicted a flow vector leading to a pixel with a similar color, thus indicating warping success. However this does not necessarily mean that the flow vector points to the correct location. For example, there could be several objects with the same color and the network could simply point to the wrong object. Letting $I_i(d_i)$ denote the pixel value in image I_i for position d_i, and $\hat{I}_i(d_i)$ denote the corresponding pixel in the image warped by the registration function, we can define the general weighting factors λ_i as:

$$\lambda_i = \frac{||I_i(d_i) - \hat{I}_i(d_i)||_2^{-1}}{\sum_j^N ||I_j(d_j) - \hat{I}_j(d_j)||_2^{-1}}. \tag{10}$$

where $\hat{I}_i = \hat{F}_{i\leftarrow t} \diamond I_t$. The MPC cost is computed as the average of the costs c_i weighted by λ_i, where each c_i is the expected distance (see equation 6) between the registered point $\hat{d}_{i,t}$ and the goal point $d_{g,i}$. Hence, the cost used for planning is $c = \sum_i \lambda_i c_i$. In the case of the single view model and a single designated pixel, the index i iterates over the start and goal image (and $N = 2$).

The proposed weighting scheme can also be used with multiple designated pixels, as used in multi-task settings and multi-view models, which are explained in section 8. The index i then also loops over the views and indices of the designated pixels.

Training Procedure The registration network is trained on the same data as the video-prediction model, but it does not share parameters with it.[3] Our approach is similar to the optic flow method proposed by [46]. However, unlike this prior work, our method computes registrations for frames that might be many time steps apart, and the goal is not to extract optic flow, but rather to determine correspondences between potentially distant images. For training, two images are sampled at random times steps t and $t + h$ along the trajectory and the images are warped to each other in both directions.

$$\hat{I}_t = \hat{F}_{t\leftarrow t+h} \diamond I_{t+h} \qquad \hat{I}_{t+h} = \hat{F}_{t+h\leftarrow t} \diamond I_t \tag{11}$$

The network, which outputs $\hat{F}_{t\leftarrow t+h}$ and $\hat{F}_{t+h\leftarrow t}$, see Figure 8 (b), is trained to minimize the photometric distance between $\hat{I}_t$ and I_t and $\hat{I}_{t+h}$ and I_{t+h}, in addition to a smoothness regularizer that penalizes abrupt changes in the outputted flow-field. The details of this loss function follow prior work [46]. We found that gradually increasing the temporal distance h between the images during training yielded better final accuracy, as it creates a learning curriculum. The temporal distance is linearly increased from 1 step to 8 steps at 20k SGD steps. In total 60k iterations were taken.

3. in principle sharing parameters with the video-prediction model might be beneficial, however this is left for future work

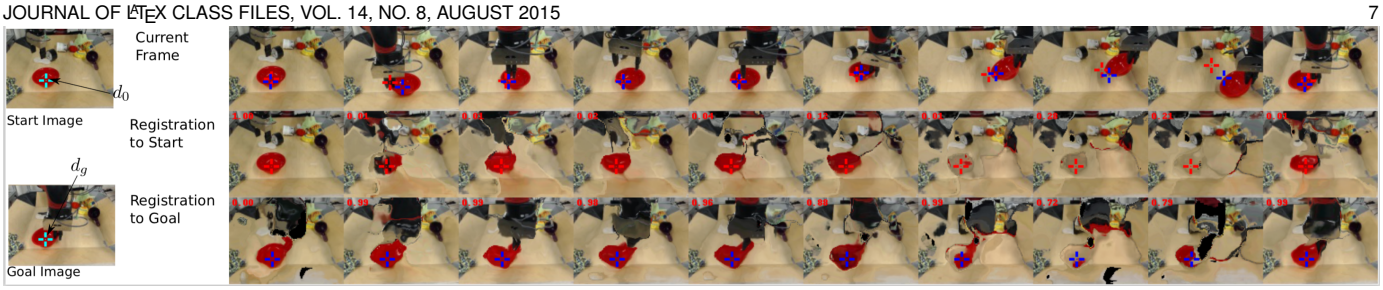


Fig. 9: Outputs of registration network. The first row shows the timesteps from left to right of a robot picking and moving a red bowl, the second row shows each image warped to the initial image via registration, and the third row shows the same for the goal image. A successful registration in this visualization would result in images that closely resemble the start- or goal image. In the first row, the locations where the designated pixel of the start image d_0 and the goal image d_g are found are marked with red and blue crosses, respectively. It can be seen that the registration to the start image (red cross) is failing in the second to last time step, while the registration to the goal image (blue cross) succeeds for all time steps. The numbers in red, in the upper left corners indicate the trade off factors λ between the views and are used as weighting factors for the planning cost. (Best viewed in PDF)

The network R is implemented as a fully convolutional network taking in two images stacked along the channel dimension. First the inputs are passed into three convolutional layers each followed by a bilinear downsampling operation. This is passed into three layers of convolution each followed by a bilinear upsampling operation (all convolutions use stride 1). By using bilinear sampling for increasing or decreasing image sizes we avoid artifacts that are caused by strided convolutions and deconvolutions.

5.3 Classifier-Based Cost Functions

An alternative way to define the cost function is with a goal classifier. This type of cost function is particularly well-suited for tasks that can be completed in multiple ways. For example, for a task of rearranging a pair objects into relative positions, i.e. pushing the first object to the left of the second object, the absolute positions of the objects do not matter nor does the arm position. A classifier-based cost function allows the planner to discover any of the possible goal states.

Unfortunately, a typical image classifier will require a large amount of labeled examples to learn, and we don't want to collect large datasets for each and every task. Instead, we aim to learn a goal classifier from only a few positive examples, using a meta-learning approach. A few positive examples of success are easy for people to provide and are the minimal information needed to convey a goal.

Formally, we consider a goal classifier $\hat{y} = f(\mathbf{o})$, where $\mathbf{o}$ denotes the image observation, and $\hat{y} \in [0, 1]$ indicates the predicted probability of the observation being of a successful outcome of the task. Our objective is to infer a classifier for a new task $\mathcal{T}_j$ from a few positive examples of success, which are easy for a user to provide and encode the minimal information needed to convey a task. In other words, given a dataset $\mathcal{D}_j^+$ of K examples of successful end states for a new task $\mathcal{T}_j$: $\mathcal{D}_j := \{(\mathbf{o}_k, 1)|k = 1...K\}_j$, our goal is to infer a classifier for task $\mathcal{T}_j$.

Meta-Learning for Few-Shot Goal Inference: To solve the above problem, we propose learning a few-shot classifier that can infer the goal of a new task from a small set of goal examples, allowing the user to define a task from a few examples of success. To train the few-shot classifier, we first collect a dataset of both positive and negative examples for a wide range of tasks. We then use this data to learn how to learn goal classifiers from a few positive examples. Our approach is illustrated in Figure 10.

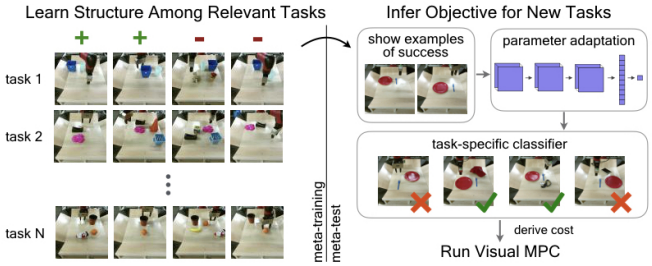


Fig. 10: We propose a framework for quickly specifying visual goals. Our goal classifier is meta-trained with positive and negative examples for diverse tasks (left), which allows it to meta-learn that some factors matter for goals (e.g., relative positions of objects), while some do not (e.g. position of the arm). At meta-test time, this classifier can learn goals for new tasks from a few examples of success (right - the goal is to place the fork to the right of the plate). The cost can be derived from the learned goal classifier for use with visual MPC.

We build upon model-agnostic meta-learning (MAML) [47], which learns initial parameters θ for model f that can efficiently adapt to a new task with one or a few steps of gradient descent. Grant et al. [48] proposed an extension of MAML, referred to as concept acquisition through meta-learning (CAML), for learning to learn new concepts from positive examples alone. We apply CAML to the setting of acquiring goal classifiers from positive examples, using a meta-training data with both positive and negative examples. The result of the meta-training procedure is an initial set of parameters that can be used to learn new goal classifiers at test time.

Test Time Procedure: At test time, the user provides a dataset $\mathcal{D}_j^+$ of K examples of successful end states for a new task $\mathcal{T}_j$: $\mathcal{D}_j := \{(\mathbf{o}_k, 1)|k = 1...K\}_j$, which are then used to infer a task-specific goal classifier C_j. In particular, the meta-learned parameters θ are updated through gradient descent to adapt to task $\mathcal{T}_j$:

$$C_j(\mathbf{o}) = f(\mathbf{o}; \theta_j') = f\big(\mathbf{o}; \theta - \alpha \nabla_\theta \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_j^+} \mathcal{L}(y_n, f(\mathbf{o}_n; \theta))\big)$$

where $\mathcal{L}$ is the cross-entropy loss function, α is the step size, and θ' denotes the parameters updated through gradient descent on task $\mathcal{T}_j$.

During planning, the learned classifier C_j takes as input an image generated by the video-prediction model and outputs the predicted probability of the goal being achieved for the task specified by the few examples of success. To convert this into a cost function, we treat the probability of success as the planning cost for that observation. To reduce the effect of false positives and mis-calibrated predictions, we use the classifier conservatively by thresholding the predictions so that reward is only given for confident successes. Below this threshold, we give a reward of 0 and above this threshold, we provide the predicted probability as the reward.

Train time procedure During meta-training, we explicitly train for the ability to infer goal classifiers for the set of training tasks, $\{\mathcal{T}_i\}$. We assume a small dataset $\mathcal{D}_i$ for each task $\mathcal{T}_i$, consisting of both positive and negative examples: $\mathcal{D}_i := \{(\mathbf{o}_n, y_n)|n = 1...N\}_i$. To learn the initial parameters θ, we optimize the following objective:

$$\min_{\theta} \sum_{i} \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_i^{\text{test}}} \mathcal{L}(y_n, f(\mathbf{o}_n; \theta_i'))$$

We optimize this objective using Adam [49] on the initial parameters θ. In our experiments, our classifier is represented by a convolutional neural network, consisting of three convolutional layers, each followed by layer normalization and a ReLU non-linearity. After the final convolutional layer, a spatial soft-argmax operation extracts spatial feature points, which are then passed through fully-connected layers.

5.4 When to use which Cost Function?

We have introduced three different kinds of cost functions, pixel-distance based cost functions with and without registration, as well as classifier-based cost functions. Here we discuss the relative strengths and weaknesses of each of them.

Pixel-distance based cost functions have the advantage that they allow moving objects precisely to target locations. Further, they do not the user to provide an image of the goal, creating a simple and easy interface for specifying goals. The pixel-distance based cost function also has a high degree of robustness against distractor objects and clutter since the optimizer can ignore the values of other pixels; this is an important feature when targeting diverse real-world environments. By incorporating an image of the goal, we can add a registration mechanism to allow for more robust closed-loop control, at the cost of a more significant burden on the user.

The classifier-based cost function allows for solving more abstract tasks where the absolute positions of an object can be irrelevant, such as position a cup in front of a plate, irrespective of where the plate is. Providing a few example images takes more effort than specifying pixel locations but allows a broader range of goal sets to be specified.

6 Trajectory Optimizer

The role of the optimizer is to find actions sequences $a_{1:T}$ which minimize the sum of the costs $c_{1:T}$ along the planning horizon T by sampling a large number of actions sequences and ranking of each video-prediction rollout using a cost function.

Algorithm 1 Trajecotry Optimization in Visual MPC

Inputs: Predictive Model g, task definition i.e. designated-pixel goal-pixel pair or goal-image
for $t = 0...T-1$ **do**
  for $i = 0...n_{iter}-1$ **do**
    if $i == 0$ **then**
      Sample M action sequences $\{a^{(m)}_{t:t+H-1}\}$ from $\mathcal{N}(0, I)$ or custom sampling distribution
    else
      Sample M action sequences $a^{(m)}_{t:t+H-1}$ from $\mathcal{N}(\mu^{(i)}, \Sigma^{(i)})$
    Use model g to predict future sequences of images $\hat{I}_{t:t+H-1}$ and probability distributions $\hat{P}_{t:t+H-1}$
    Rank the action sequences using a cost function c
    Fit a Gaussian to the k best action samples yielding $\mu^{(i)}, \Sigma^{(i)}$

To render the planning process more efficient, we use the cross-entropy method (CEM), a gradient-free optimization procedure. CEM consists of iteratively resampling action sequences and refitting Gaussian distributions to the actions with the best predicted cost. We extend CEM to handle a mixture of continuous and discrete actions.

The motivation for using a gradient-free, sampling-based optimizer is that in this way we can easily ensure that actions stay within the distribution of actions the model has encountered during training. This is crucial to ensure that the model does not receive out-of-distribution inputs and makes valid predictions.

In the appendix A we explain a few improvements we made to the CEM-optimizer.

7 Custom Action Sampling Distributions

When collecting data by sampling from simple distributions, such as a multivariate Gaussian, the skills that emerged were found to be generally restricted to pushing and dragging objects. This is because with simple distributions, it is very unlikely to visit states like picking up and placing of objects or folding cloth. Not only would the model be imprecise for these kinds of states, but also during planning it would be unlikely to *find* action sequences that grasp an object or fold a piece of cloth. We therefore explore how the sampling distribution used both in data collection and sampling-based planning can be changed to visit these, otherwise unlikely, states more frequently, allowing more complex behavior to emerge.

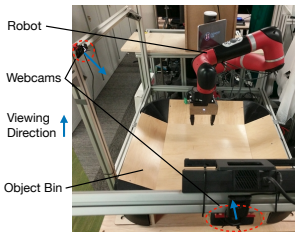


Fig. 11: Robot setup, with 2 standard web-cams arranged at different viewing angles.

Learning Pick and Place Behavior: We first discuss picking and placing of objects. To allow picking up and placing of objects to occur more frequently, we incorporate a simple

"reflex" during data collection, where the gripper automatically closes, when the height of the wrist above the table is lower than a small threshold. This reflex is inspired by the palmar reflex observed in infants [50]. With this primitive, about 20% of training trajectories included some sort of grasp on an object. It is worth noting that, other than this reflex, no grasping-specific engineering was applied to the policy, allowing a joint pushing and grasping policy to emerge through planning (see figure 16 in the appendix). In our experiments, we evaluate our method using data obtained both with and without the grasping reflex, evaluating both purely non-prehensile and combined prehensile and non-prehensile manipulation.

Learning Deformable Object Manipulation: In order to collect meaningful interaction data for learning folding of deformable objects such as towels and cloths, we adapted the sampling distribution to increase the likelihood of encountering "cloth-folding-states" in the data. When using actions sampled from a simple distribution or the previously-described distribution, cloths would become tangled and we would see many "sweeping motions". To improve the efficiency of encountering "cloth folds", we use the same action primitive used both for grasping primitive, but additionally we reduce lateral motion of the end-effector when the gripper is close to the table, thus reducing the undesired "sweeping motions".

Further details about the data collection process can be found in section B in the appendix.

8 Multi-View Visual MPC

The visual MPC algorithm as described so far is only able to solve manipulation tasks specified in 2D, like rearranging objects on the table. However, this can impose severe limitations; for example, a task such as lifting an object to a particular position in 3D cannot be fully specified with a single view, since it would be ambiguous.

We use a combination of two views, taken from two cameras arranged appropriately, to jointly define a 3D task. Figure 11 shows the robot setup, including two standard webcams observing the workspace from different angles. The registration method described in the previous section is used separately per view to allow for dynamic retrying and solving temporally extended tasks. The planning costs from each view are combined using weighted averaging where the weights are provided by the registration network (see equation 10). The the first two rows of figure 14 show a "placing task" specified in two views, where an object needs

9 Experimental Evaluation

In this section we present both qualitative and quantitative performance evaluations of visual MPC on various manipulation tasks assessing the degree of generalization and comparing different prediction models and cost functions and with a hand-crafted baseline.

Qualitative Evaluation In figures 1 and 14 we present a set of qualitative experiments showing that visual MPC trained fully self-supervised is capable of solving a wide range of complex tasks. Videos for the qualitative examples are at the following webpage[4]

4. https://sites.google.com/view/visualforesight/

	moved imp. ± std err. of mean	stationary imp. ± std err. of mean
DNA [5]	0.83 ±0.25	-1.1 ± 0.2
SNA	**10.6 ± 0.82**	**-1.5 ± 0.2**

TABLE 1: Results for multi-objective pushing on 8 object/goal configurations with 2 seen and 2 novel objects. Values indicate improvement in distance from starting position, higher is better. Units are pixels in the 64x64 images.

Quantitative Evaluation In order to perform quantitative comparisons, we define a set of benchmark tasks where the robot is required to move object(s) into a goal configuration. For measuring success, we use a distance-based evaluation where a human annotates the positions of the objects after pushing allowing us to compute the remaining distance to the goal.

9.1 Comparing Video Prediction Architectures

We first aim to answer the question: Does visual MPC using the occlusion-aware SNA video prediction model that includes temporal skip connections outperform visual MPC with the dynamic neural advection model (DNA) [5] *without* temporal skip-connections? We evaluate on multi-objective tasks where one object must be pushed without disturbing another.

To examine whether our skip-connection model (SNA) helps with handling occlusions, we devised a task that requires the robot to push one object, while keeping another object stationary. When the stationary object is in the way, the robot must move the target object around it. This is illustrated on the left side of Figure 17 in the appendix. While pushing the target object, the gripper may occlude the stationary object, and the task can only be performed successfully if the model can make accurate predictions through this occlusion. These tasks are specified by selecting one starting pixel on the target object, one goal pixel location for the target object, and one pixel location on the obstacle used both as the start and the goal to indicate that it should not be moved.

We use four different object arrangements with two training objects and two objects that were not seen during training. We find that, in most cases, the SNA model is able to find a valid trajectory, while the DNA model, that is not able to handle occlusion, is mostly unable to find a solution. The results of our quantitative comparisons are shown in Table 1, indicating that temporal skip-connections indeed help with handling occlusion in combined pushing and obstacle avoidance tasks.

9.2 Evaluating Registration-based Cost Functions

In this section we ask: How important is it to update the model's belief of where the target objects currently are? We first provide two qualitative examples:

In example (1) of figure 1 the task is to bring the stuffed animal to a particular location in 3D-space on the other side of the arena. To test the system's reaction to perturbations that could be encountered in open-world settings, during execution a human knocks the object out of the robot's hand.

	Short	Long
Visual MPC + predictor propagation	83%	20%
Visual MPC + OpenCV tracking	83%	45%
Visual MPC + registration network	83%	**66%**

TABLE 2: Success rate for long-distance pushing benchmark with 20 different object/goal configurations and short-distance benchmark with 15 object/goal configurations. Success is defined as bringing the object closer than 15 pixels to the goal, which corresponds to around $7.5cm$.

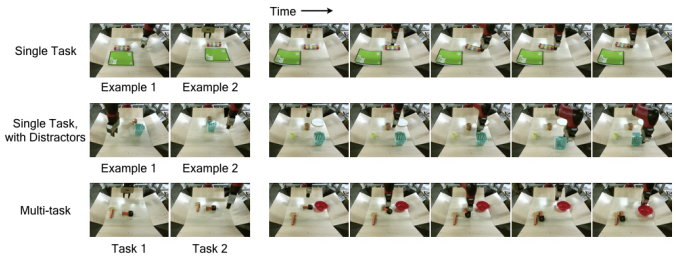


Fig. 12: Object arrangement performance of our goal classifier with distractor objects and with two tasks. The left shows a subset of the 5 positive examples that are provided for inferring the goal classifier(s), while the right shows the robot executing the specified task(s) via visual planning.

The experiment shows that visual MPC is able to naturally perform a new grasp attempt and bring the object to the goal.

In figure 15 in the appendix, the task is to push the bottle to point marked with the green dot. In this the system recovers from an initial failure.

The next question we investigate is: How much does this matter for short horizon versus long horizon tasks? In this experiment, we disable the gripper control, which requires the robot to push objects to the target. We compare two variants of updating the positions of the designated pixel when using a pixel-distance based cost function. The first is a cost function that uses a registration-based method, trained in a fully self-supervised fashion, and the second is with a cost function that uses off-the shelf tracking from OpenCV [51]. Additionally we compare to visual MPC that uses the video-prediction model's own prior predictions to update the current position of the designated pixel, rather than tracking the object with registration or tracking.

We evaluate our method on 20 long-distance and 15 short-distance pushing tasks. For long distance tasks the initial distance between the object and its goal-position is $30cm$ while for short distance tasks it is $15cm$. Table 2 lists quantitative comparisons showing that on the long distance benchmark visual MPC using the registration-based cost not only outperforms prior work [6], but also outperforms the hand-designed, supervised object tracker [51]. By contrast for the short distance benchmark, all methods perform comparably. Thus, theses results demonstrate the importance of closed loop control for *long-horizon tasks*, while for short-horizon tasks object tracking appears to be irrelevant.

9.3 Evaluating Classifier-based Cost Function

The goal of the classifier-based cost function is to provide an easy way to compute an objective for new tasks from a few observations of success for that task, so we compare our approach to alternative and prior methods for doing so under the same assumptions: pixel distance and latent space distance. In the latter, we measure the distance between the current and goal observations in a learned latent space, obtained by training an autoencoder (DSAE) [52] on the same data used for our classifier. Because we are considering a different form of task specification, we do not compare the classifier-based cost function to costs based on user-designated pixels or registration.

To collect data for meta-training the classifier, we randomly select a pair of objects from our set of training objects, and position them into many different relative positions, recording the image for each configuration. One task corresponds to a particular relative positioning of two objects, e.g. the first object to the left of the second, and we construct positive and negative examples for this task by labeling the aforementioned images. We randomly position the arm in each image, as it is not a determiner of task success. A good objective should ignore the position of the arm. We also include randomly-positioned distractor objects in about a third of the collected images.

We evaluate the classifier-based cost function in three different experimental settings. In the first setting, the goal is to arrange two objects into a specified relative arrangement. The second setting is the same, but with distractor objects present. In the final, most challenging setting, the goal is to achieve two tasks in sequence. We provide positive examples for both tasks, infer the classifier for both task, perform MPC for the first task until completion, followed by MPC for the second task. To evaluate the ability to generalize to new goals and settings, we use novel, held-out objects for all of the task and distractor objects in our evaluation.

We qualitatively visualize the benchmark tasks in Figure 12. On the left, we show a subset of the five images provided to illustrate the task(s), and on the left, we show the motions performed by the robot. We see that the robot is able to execute motions which lead to a correct relative positioning of the objects.

We quantitatively evaluate each method across 20 tasks, including 10 unique object pairs. The results, shown in Figure 13, indicate that the distance-based metrics struggle to infer the goal of the task, while our approach leads to substantially more successful behavior on average. TO-DO: explain how exactly you measure success

9.4 Evaluating Multi-Task Performance

One of the key motivations for visual MPC is to build a system that can solve a *wide variety* of different tasks, involving completely different objects, physics and semantics. Examples for tasks that can be solved with visual MPC are shown in figure 1 and 14. The tasks (1) and (2) are object rearrangement tasks, in task (3) a towel needs to be wrapped around an object while keeping the object stationary. The example shown in (4) and all examples in figure 12 show relative object rearrangement tasks. Examples (5) and (6) show the same "placing task" where an object needs to be grasped and placed onto a plate. In (7) the task is to move the black object to the goal location while avoiding the obstacle in the middle which is marked with a designated- and goal-pixel.

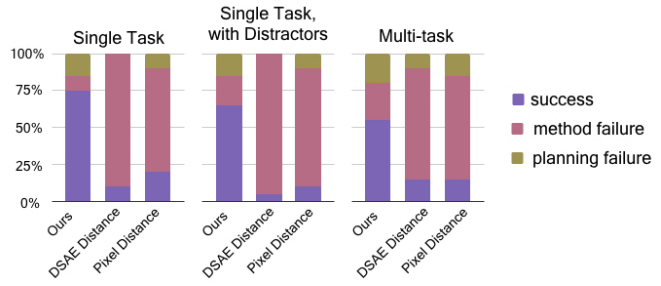


Fig. 13: Quantitative performance of visual planning for object rearrangement tasks across different goal specification methods: our meta-learned classifier, DSAE [52], and pixel error. Where possible, we include break down the cause of failures into errors caused by inaccurate prediction or planning and those caused by an inaccurate goal classifier.

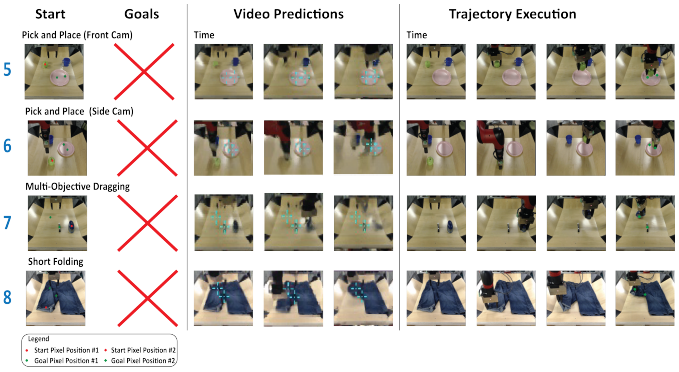


Fig. 14: Visual MPC successfully solves a wide variety of tasks including multi-objective tasks, such as placing an object on a plate (row 5 and 6), object positioning with obstacle avoidance (row 7) and folding shorts (row 8).

	% of Trials with Final Pixel Distance < 15
Visual MPC	**75%**
Calibrated Camera Baseline	18.75 %

TABLE 3: Results for a combined benchmark of 10 hard object pushing and grasping tasks, along with 6 cloth folding tasks. Values indicate the percentage of trials which ended with the object closer than a threshold distance (measured in pixels) to the designated goal. Higher is better.

The generality of visual MPC mainly stems from two components — the generality of the visual dynamics model and the generality of the task definition. We found that the dynamics model often generalizes well to objects outside of the training set, if they have similar properties to the objects it was trained with. For example, trajectory (4) in figure 14 shows the model predicting a pair of shorts being folded, although the model never encountered shorts during training, towels and shirts were the only cloths that were part of the training set. It is worth noting that, in all qualitative examples, the predictions are performed by the same model for objects not part of the training set. The second component allowing for generality is the high degree of flexibility for task definition. Using designated pixels object positioning tasks can be defined in 3D space, as shown examples (1) and (2) in figure and examples (3)-(5) in figure 14, when adding a goal-image the positioning accuracy can be improved as thanks to the presented registration scheme. For tasks where we care about *relative* rather than absolute positioning a meta-learned classifier can be used.

Next we present a benchmark to quantitatively answer the following question: How does visual MPC compare to a hand-engineered baseline on a large number of diverse tasks?

Hand-crafted Baseline For this comparison we engineered a simple trajectory generator to perform a grasp at the location of the initial designated pixel, lift the arm and bring it to the position of the goal-pixel. A camera calibration was performed to carry out the necessary conversions between image-space and robot work-space coordinates. For simplicity, the baseline controller executes open-loop. Therefore to allow for a fair comparison visual-MPC is also executed open-loop, i.e. no registration or tracking is used.

Altogether we selected 16 tasks, some of them being identical to the qualitative examples presented earlier. The results of the benchmark, shown in table 3, indicate that visual MPC clearly outperforms the baseline. Visual MPC succeeded for most of the benchmark tasks. While the baseline succeeded for some of the cloth folding tasks, but failed for almost all of the object relocation tasks. We attribute this to the fact that it does not reason about physics, therefore it is lacking the capability of positioning the end-effector in a location that is appropriate for grasping or pushing.

Discussion Generalization and multi-task performance is arguably one of the biggest challenges in robotic manipulation today. While deep learning has relieved us from much of the problem-specific engineering, most of the works either require extensive amounts of labeled data or in the case of deep reinforcement learning, focus on the mastery of single tasks while relying on human-provided reward signals. From the experiments with visual MPC, especially the qualitative examples and the multi-task benchmark, we can conclude that visual MPC *generalizes* to a wide range of tasks it has never seen during training. This is in contrast to many model-free approaches for robotic control which often struggle to perform well on novel tasks. Most of the generalization performance is likely a result of large-scale self-supervised learning, which allows to acquire a rich dynamics model of the environment.

10 Conclusion

We presented an algorithm that leverages self-supervision from visual prediction to learn a deep dynamics model on images, and show how it can be embedded into a planning framework to solve a variety of robotic control tasks. We demonstrate that visual model-predictive control is able to successfully perform multi-object manipulation, pushing, picking and placing, and cloth-folding tasks – all within a single framework. These tasks involve complex contact dynamics as well as deformable objects, indicating that the same sensory prediction model can be used to capture a wide variety of environment dynamics with enough accuracy to allow for control.

Limitations The main limitations of the presented framework are that all target objects need to be visible throughout

execution, it is currently not possible to handle partially observed domains. This is especially important for tasks that require objects to be brought into occlusion (or taken out of occlusion), for example putting an object in a box and closing it. Another limitation is that the tasks are still of only medium duration and usually only touch one or two objects. Longer-term planning remains an open problem. Lastly, the fidelity of object positioning is still significantly below what humans can achieve.

Possible future directions The key advantage of a model-based deep-reinforcement learning algorithm like visual MPC is that it *generalizes* to tasks it has never encountered before. This makes visual MPC a good candidate for a building block of future robotic manipulation systems that will be able solve an even wider range of complex tasks with much longer horizons.

REFERENCES

[1] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end learning of deep visuomotor policies," *Journal of Machine Learning Research (JMLR)*, 2016.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[5] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 2017.

[6] F. Ebert, C. Finn, A. X. Lee, and S. Levine, "Self-supervised visual planning with temporal skip connections," *CoRR*, vol. abs/1710.05268, 2017. [Online]. Available: http://arxiv.org/abs/1710.05268

[7] F. Ebert, S. Dasari, A. X. Lee, S. Levine, and C. Finn, "Robustness via retrying: Closed-loop robotic manipulation with self-supervised learning," *arXiv preprint arXiv:1810.03043*, 2018.

[8] A. Xie, A. Singh, S. Levine, and C. Finn, "Few-shot goal inference for visuomotor learning and planning," *arXiv preprint arXiv:1810.00482*, 2018.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[10] R. S. Sutton, A. G. Barto, F. Bach *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[11] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, 2005, pp. 1281–1288.

[12] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International Conference on Machine Learning (ICML)*, vol. 2017, 2017.

[13] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[14] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *arXiv preprint arXiv:1805.12114*, 2018.

[15] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[16] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[17] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," *arXiv preprint arXiv:1708.02596*, 2017.

[18] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *Advances in neural information processing systems*, 2015, pp. 2746–2754.

[19] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," *International Conference on Learning Representations (ICLR)*, 2017.

[20] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[21] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in atari games," in *Advances in Neural Information Processing Systems*, 2015.

[22] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine, "Solar: Deep structured latent representations for model-based reinforcement learning," *arXiv preprint arXiv:1808.09105*, 2018.

[23] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Advances in Neural Information Processing Systems*, 2016, pp. 5074–5082.

[24] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine, "Combining self-supervised learning and imitation for vision-based rope manipulation," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2146–2153.

[25] A. Nair, P. Agarwal, D. Chen, P. Isola, P. Abbeel, and S. Levine, "Combining self-supervised learning and imitation for vision-based rope manipulation," *International Conference on Robotics and Automation (ICRA)*, 2017.

[26] R. Mottaghi, M. Rastegari, A. Gupta, and A. Farhadi, "what happens if... learning to predict the effect of forces in images," in *European Conference on Computer Vision*. Springer, 2016, pp. 269–285.

[27] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *International Conference on Robotics and Automation (ICRA)*, 2016.

[28] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *International Journal of Robotics Research (IJRR)*, 2016.

[29] R. Calandra, A. Owens, M. Upadhyaya, W. Yuan, J. Lin, E. H. Adelson, and S. Levine, "The feeling of success: Does touch sensing help predict grasp outcomes?" *arXiv preprint arXiv:1710.05512*, 2017.

[30] L. Pinto, D. Gandhi, Y. Han, Y.-L. Park, and A. Gupta, "The curious robot: Learning visual representations via physical interactions," in *European Conference on Computer Vision*. Springer, 2016, pp. 3–18.

[31] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[32] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," *arXiv preprint arXiv:1704.05588*, 2017.

[33] S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed, "Recurrent environment simulators," *arXiv preprint arXiv:1704.02254*, 2017.

[34] B. Boots, A. Byravan, and D. Fox, "Learning predictive models of a depth camera & manipulator from raw execution traces," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.

[35] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Neural Information Processing Systems (NIPS)*, 2016.

[36] N. Kalchbrenner, A. v. d. Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Video pixel networks," *arXiv preprint arXiv:1610.00527*, 2016.

[37] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," *International Conference on Learning Representations (ICLR)*, 2016.

[38] T. Kurutach, A. Tamar, G. Yang, S. Russell, and P. Abbeel, "Learning plannable representations with causal infogan," *arXiv preprint arXiv:1807.09341*, 2018.

[39] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, 2015.

[40] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *Advances In Neural Information Processing Systems*, 2016.
[41] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *International Conference on Learning Representations (ICLR)*, 2017.
[42] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool, "Dynamic filter networks," in *Neural Information Processing Systems (NIPS)*, 2016.
[43] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," *arXiv preprint arXiv:1606.02378*, 2016.
[44] S. Reed, A. v. d. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, D. Belov, and N. de Freitas, "Parallel multiscale autoregressive density estimation," *arXiv preprint arXiv:1703.03664*, 2017.
[45] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, "View synthesis by appearance flow," in *European conference on computer vision*. Springer, 2016, pp. 286–301.
[46] S. Meister, J. Hur, and S. Roth, "Unflow: Unsupervised learning of optical flow with a bidirectional census loss," *arXiv preprint arXiv:1711.07837*, 2017.
[47] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *International Conference on Machine Learning (ICML)*, 2017.
[48] E. Grant, C. Finn, J. Peterson, J. Abbott, S. Levine, T. Griffiths, and T. Darrell, "Concept acquisition via meta-learning: Few-shot learning from positive examples," in *NIPS Workshop on Cognitively-Informed Artificial Intelligence*, 2017.
[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.
[50] D. Sherer, "Fetal grasping at 16 weeks' gestation," *Journal of ultrasound in medicine*, vol. 12, no. 6, 1993.
[51] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 983–990.
[52] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *International Conference on Robotics and Automation (ICRA)*, 2016.
[53] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

Frederik Ebert Frederik Ebert received a BS in Mechatronics and Information Technology as well a MS in "Robotics, Cognition, Intelligence (RCI)" from the Technical University of Munich (TUM). He is currently a PhD student at Berkeley Artifical Intelligence Research (BAIR).

Chelsea Finn Chelsea Finn is currently a research scientist at Google Brain and a post-doctoral scholar at UC Berkeley, and will join the Computer Science faculty at Stanford University in 2019.

Annie Xie Annie Xie is pursuing a B.S. degree in Electrical Engineering and Computer Science at UC Berkeley.

Sudeep Dasari Sudeep Dasari is a 4th year student at UC Berkeley pursuing a B.S in Electrical Engineering and Computer Science.

Alex Lee Alex Lee received a BS in Electrical Engineering and Computer Science from UC Berkeley in 2013, and he is currently pursuing a PhD in Computer Science from UC Berkeley.

Sergey Levine Sergey Levine received a BS and MS in Computer Science from Stanford University in 2009, and a Ph.D. in Computer Science from Stanford University in 2014. He joined the faculty of the Department of Electrical Engineering and Computer Sciences at UC Berkeley in fall 2016.

APPENDIX A
IMPROVEMENTS OF THE CEM-OPTIMIZER

In the model-predictive control setting, the action sequences found by the optimizer can be very different between execution real-world times steps. For example at one time step the optimizer might find a pushing action leading towards the goal and in the next time step it determines a grasping action to be optimal to reach the goal. Naïve replanning at every time step can then result in alternating between a pushing and a grasping attempt indefinitely causing the agent to get stuck and not making any progress towards to goal.

We can resolve this problem by modifying the sampling distribution of the first iteration of CEM so that the optimizer commits to the plan found in the previous time step. In the simplest version of CEM the sampling distribution at first iteration of CEM is chosen to be a Gaussian with diagonal covariance matrix and zero mean. We instead use the best action sequence found in the optimization of the *previous* real-world time step as the mean for sampling new actions in the *current* real-world time-step. Since this action sequence is optimized for the previous time step we only use the values $a_{2:T}$ and omit the first action. To sample actions close to the action sequence from the previous time step we reduce the entries of the diagonal covariance matrix for the first $T-1$ time steps. It is crucial that the last entry of the covariance matrix at the end of the horizon is not reduced otherwise no exploration could happen for the last time step causing poor performance at later time steps.

APPENDIX B
EXPERIMENTAL SETUP

To train both our video-prediction and registration models, we collected 20,000 trajectories of pushing motions and 15,000 trajectories with gripper control, where the robot randomly picks and moves objects using the grasping reflex described in section 7. The data collection process is fully autonomous, requiring human intervention only to change out the objects in front of the robot. The action space consisted of relative movements of the end-effector in cartesian space along the x, y, and z axes, and for some parts of the dataset we also added azimuthal rotations of the gripper and a grasping action.

APPENDIX C
EXPERIMENTAL EVALUATION

C.1 Experiments with multiple-objects rearrangement with occlusion

. Figure 17 TO-DO: change thisshows an example of the SNA model successfully predicting the position of the obstacle through an occlusion and finding a trajectory that avoids the obstacle.

APPENDIX D
SIMULATED EXPERIMENTS

In order to provide a more controlled comparison, we also set up a realistic simulation environment using MuJoCo

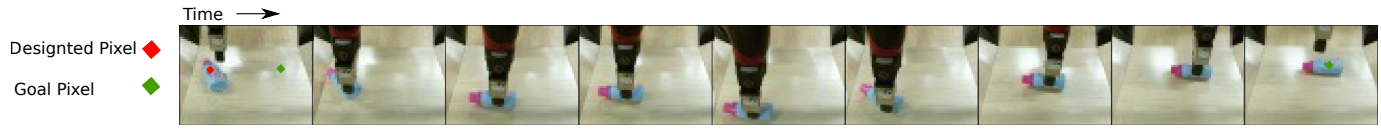


Fig. 15: Applying our method to a pushing task. In the first 3 time instants the object behaves unexpectedly, moving down. The tracking then allows the robot to retry, allowing it to eventually bring the object to the goal.

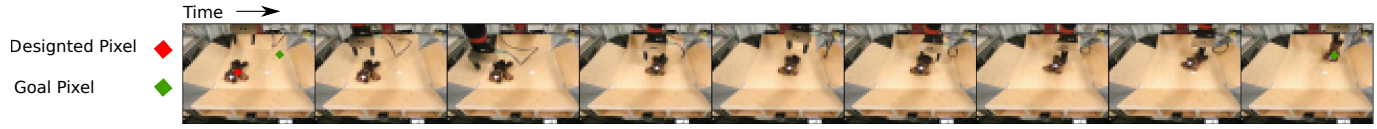


Fig. 16: Retrying behavior of our method combining prehensile and non-prehensile manipulation. In the first 4 time instants shown the robot pushes the object. It then loses the object, and decides to grasp it pulling it all the way to the goal. Retrying is enabled by applying the learned registration to both camera views (here we only show the front view).

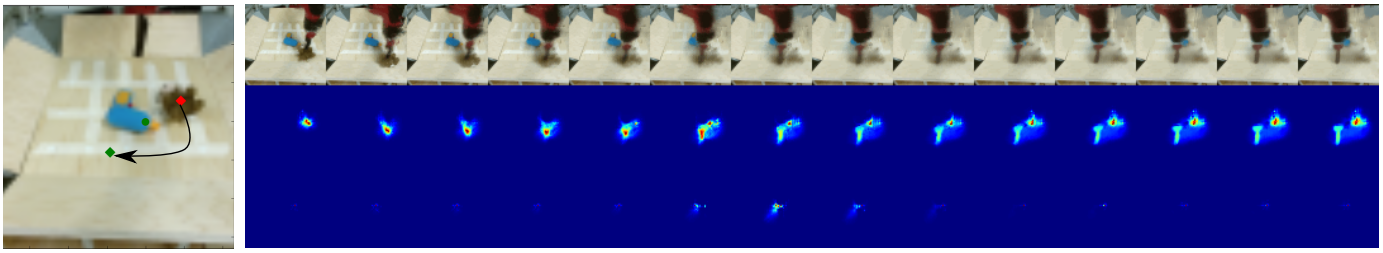


Fig. 17: Left: Task setup with green dot marking the obstacle. Right, first row: the predicted frames generated by SNA. Second row: the probability distribution of the designated pixel on the *moving* object (brown stuffed animal). Note that part of the distribution shifts down and left, which is the indicated goal. Third row: the probability distribution of the designated pixel on the obstacle-object (blue power drill). Although the distribution increases in entropy during the occlusion (in the middle), it then recovers and remains on its original position.

[53], which includes a robotic manipulator controlled via Cartesian position control, similar to our real world setup, pushing randomly-generated L-shaped objects with random colors (see details in supplementary materials). We trained the same video prediction model in this environment, and set up 50 evaluation tasks where blocks must be pushed to target locations with maximum episode lengths of 120 steps. We compare our proposed registration-based method, "predictor propagation," and ground-truth registration obtained from the simulator, which provides an oracle upper bound on registration performance. **??** shows the results of this simulated evaluation, where the x-axis shows different distance thresholds, and the y-axis shows the fraction of evaluation scenarios where each method pushed the object within that threshold. We can see that, for thresholds around 0.1, our method drastically outperforms predictor propagation (i.e., prior work [6]), and has a relatively modest gap in performance against ground-truth tracking. This indicates that our registration method is highly effective in guiding visual MPC, despite being entirely self-supervised.