Visual MPC: 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 algorithms capable of learning complex skills directly from raw sensory inputs have the potential to allieviate the need for hand-designing features and control laws for autnomomous systems such as robot manipulators and or self-driving vehicles. However, deep reinforcement learning for real-world robotic control has yet to achieve the kind of generalization and broad applicability demonstrated by deep learning algorithms in other domains, such as vision and natural language processing. We present a deep reinforcement learning method that is practical for real-world robotics tasks, such as robotic manipulation, and can generalize effectively to never-before-seen tasks and objects, without requiring meticulously hand-specified reward functions and carefully engineered episodic learning setups. To that end, we opt for a self-supervised approach based on *prediction*: instead of learning to directly maximize a given reward, our method is based on learning to predict the future, directly in terms of the raw sensory observations of the robot, such as camera images. Beyond training a visual predictive model that can make predictions that are accurate enough for real-world control, a practical implementation of this approach requires us to address the challenge of task specification. Ground truth reward signals are generally unavailable in the real world, therefore we explore three distinct methods for goal specification for visual model-based RL for robotic manipulation: designated pixels, where a user specifies desired object relocation tasks by selecting particular points in an image and corresponding goal positions, goal images, where the desired goal state is specified with an image, and tasks defined by image classifiers. Our deep predictive models are trained using data collected entirely autonomously by a robot interacting with hundreds of objects. 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. TO-DO: Page limit is 12!!

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, remarkably, are able to learn a range of complex yet generalizable concepts 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, open-world 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, and, crucially, prediction of raw sensory observations does not make any assumptions about the availability of state information or extrinsic reward feedback. Learning via prediction is possible directly from a stream of raw sensory observations, such as images from a camera. These observations are both information-rich and high-dimensional, presenting both an opportunity and a challenge. Future observations provide a substantial amount

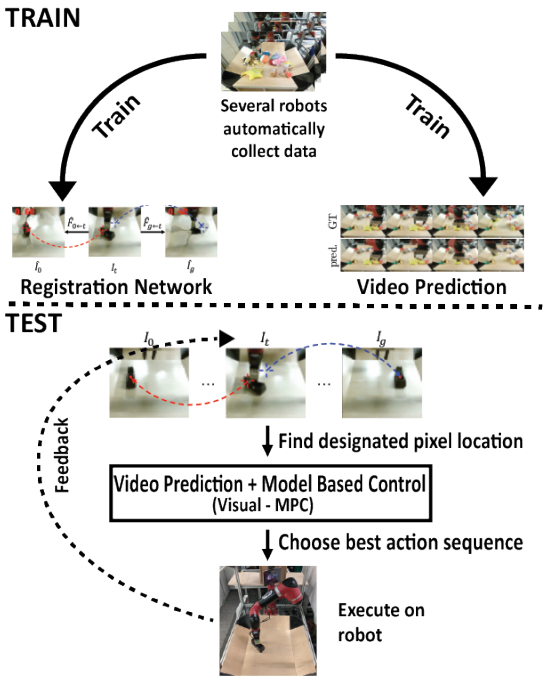


Fig. 1: TO-DO: Example Trajectories for different tasks, solved by Visual MPC

- *The first two authors contributed equally.*

Manuscript received xx.xx.xxx

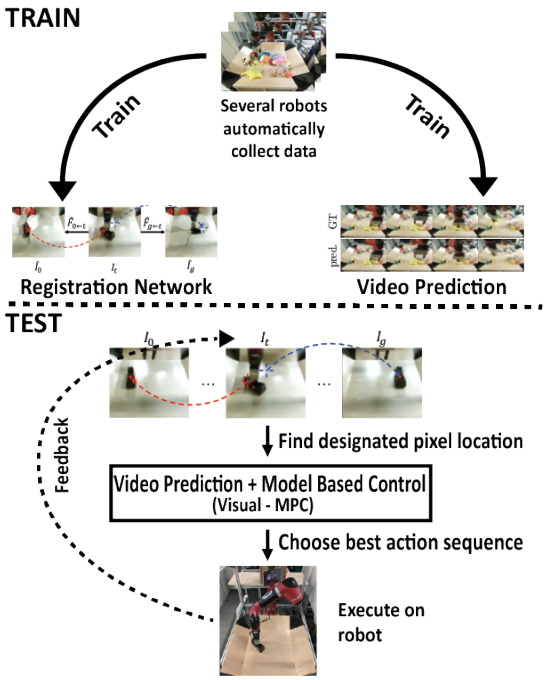


Fig. 2: TO-DO: Overview of visual MPC concept

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.

To study *generalization* in reinforcement learning, we must enable our model to learn from a diverse range of interactions with varied objects. We therefore consider a minimally structured robotic control domain, where data is collected by the robot via unsupervised interaction with a wide range of objects. 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, in order to study learning in diverse settings, necessary in order to enable and 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, and a predictive model can be used at test time to perform new tasks without additional learning. We therefore propose to learn action-conditioned predictive models directly on raw pixel observations, and show that they can be used to accomplish manipulation tasks on a real robot in the physical world at test-time. Sensory prediction models are goal-agnostic, which enables learning for a variety of different goals at the same time. By learning from raw sensor readings, they are also fully general, in that they do not require access to any other state representation beyond the observations. Our overall approach amounts to a deep model-based reinforcement learning algorithm that leverages video prediction models to perform a variety of pixel-based control tasks.

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. 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, and 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.

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, analysis of each method involving XX, as well as a comprehensive, open-sourced simulation environment to facilitate future research and better reproducibility.

TO-DO: make big VMPC diagramm!!

2 Related Work

Learning from rewards. Reinforcement learning typically assumes access to an external reward signal, effectively *reinforcing* 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* cost functions. These internal costs are computed based on rollouts of the sensory prediction model, as detailed in section 5.

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 [11], [12], and have been used with both low-dimensional [13] and high-dimensional [14] 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 [15] and video game environments [16], 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 [17]. 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.

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 [18], [19], [20] or obstacle avoidance [21], [22]. reviewers. Also cite Roberto's paper probably. 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 or reward function during data collection.

Prior work has also used "inverse models," which learn to predict actions that lead from one state to another state specified by sensory inputs, to learn skills such as object pushing [23] and rope tying [24]. Pinto et al. [25] perform self-supervised learning in a multi-task setting combing learning an inverse model for pushing, a grasp success metric and pose-invariant image embeddings. However, such methods struggle when applied to more extended or continuous tasks, where their inability to support forward planning and tendency to pick up on extraneous distractors (e.g., predicting the action just from observing the arm) present severe challenges. We observe a substantial improvement in the length and complexity of manipulations that can be performed with our method.

Kurutach et al. use a InfoGAN model [26] to learn a latent-space describing the environment states in which planning can be performed. The model can be used to obtain a sequence of waypoints in the latent space between the current state and goal-state. The method currently relies on an inverse model to reach the states proposed by the InfoGAN model. So far experiments have been limited to rope rearrangement tasks.

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 synthetic video game images [27], [28] and robotic manipulation [29], [30], [31]. Video prediction without actions has been studied for unstructured videos [32], [33], [34] and driving [35], [36]. Several works have sought to use more complex distributions for future images, for example by using autoregressive models [31], [37]. While this often produces sharp predictions, the resulting models are extremely demanding computationally which would be impractical for real-world robotic control. In this work, we extend video prediction methods that are based on predicting a transformation from the previous image [30], [36]. Prior work has also sought to predict motion directly in 3D, using 3D point clouds obtained from a depth camera [38], requiring point-to-point correspondences over time, which makes it hard to apply to previously unseen objects. Our predictive model is effective for a wide range of real-world object manipulations and does not require 3D depth sensing or point-to-point correspondences between frames. Crucially, we demonstrate that our models enable real-world robotic control, making it possible to perform a variety of user-specified manipulation tasks.

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 fully autonomously by applying random actions sampled from a specific distribution. It is crucial that this distribution allows to frequently visit the parts of the state space which are relevant for solving the intended tasks. Therefore we design a custom sampling distribution, directing data collection to the desired parts of the state space, as further detailed in section 7 and 8.4.

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, 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.

In order to correct for mistakes made by the model, the actions are iteratively replanned at each real-world time step[1] $\tau \in \{0, ..., \tau_{max}\}$ following the framework of model-predictive control (MPC): at each real-world step τ, 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 the next section the video-prediction model will be explained, followed by a description of 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 [30]. 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

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

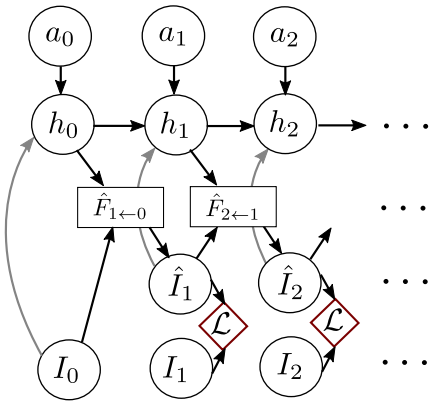


Fig. 3: Simplified structure of the video-prediction model. Time goes from left to right, a_t are the actions, h_t are the hidden states in the recurrent neural network, $\hat{F}_{t+1\leftarrow t}$ is a 2D-warping field, I_t are real images, and $\hat{I}_t$ are predicted images, $\mathcal{L}$ is a pairwise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

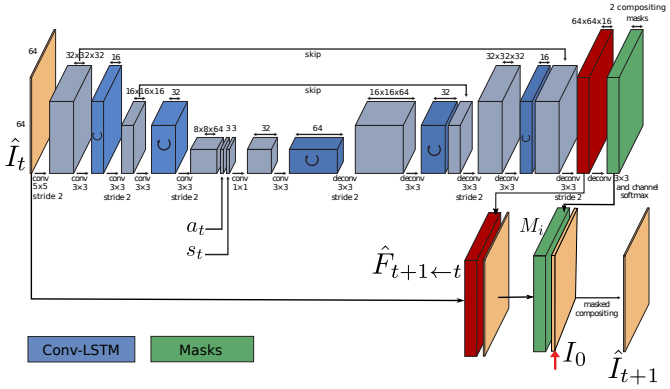


Fig. 4: Forward pass through the recurrent SNA model based on Equation (5). The red arrow indicates where the image from the first time step I_0 is concatenated with the transformed images $\hat{F}_{t+1\leftarrow t} \diamond \hat{I}_t$ multiplying each channel with a separate mask to produce the predicted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

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 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 [39]. 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

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

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 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 7. 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 7(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 7(a).

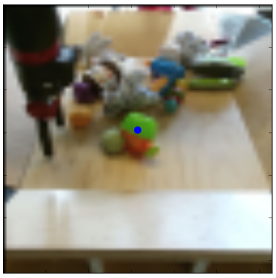


Fig. 5: The blue dot indicates the designated pixel

TO-DO: can cut from here: Next, we show another illustration comparing the occlusion handling of DNA and the proposed SNA model. The graphs in Figure 6 show the predicted probability evaluated at the position of the designated pixel, shown in figure 5, which is stationary during the entire motion. Precisely when the arm occludes the designated pixel, the probability at this point decreases. This indicates that the model is 'unsure' where this pixel is. When the arm unoccludes the designated pixel, it should become visible again, and the probability of the designated pixel being at its original position should go up. In the case of the DNA model and its variants [30], the probability mass does not increase after the object reappears.

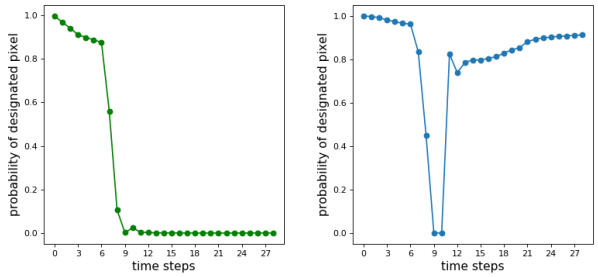
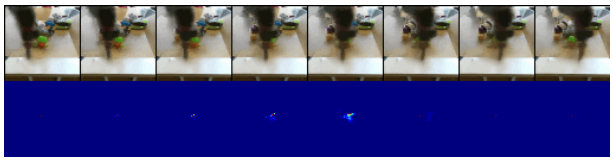
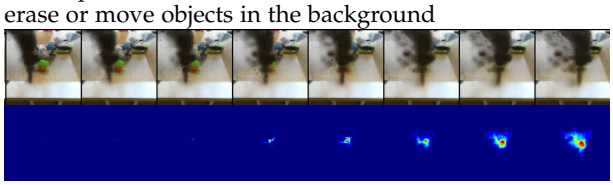


Fig. 6: Predicted probability $P_{d^{(0)}}(t)$ of the designated pixel being at the location of the blue dot indicated in Figure 5 for the DNA model (left) and the SNA model (right). TO-DO: can cut if no more space!



(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. 7: 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.

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 8.

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:

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

The per time-step costs c_t are summed together giving the overall planing objective c. An alternative choice for the cost

function is to evaluate P_t at the position of the goal-point, which describes as how likely the model predicts an action sequence will hit the goal *exactly*, we call this method *goal-point evaluation*:

$$c = \sum_{t=1,\ldots,T} P_t(g) \tag{7}$$

The expected distance to the goal provides a smoother planning objective than the goal-point evaluation cost 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 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 two simple alternative approaches 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 are a number of issues with this approach: first, 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. Second, due to the blurry predictions from a video prediction model, the pixel-wise difference between the predictions and the goal image can become meaningless.

Another approach is to perform a registration between predicted video frames and the goal image, and use the average length of the warping vectors as a cost function for "closeness" to the goal image. However, a major drawback of cost functions based on metrics computed on the complete image is that they naturally emphasize large objects (such as the robot's arm), while small objects only contribute negligible amounts to the costs. As a result, the planner only tries to match the positions of the large objects (the arm), ignoring smaller objects.

Test Time Procedure We will first describe the registration scheme at test time (see Figure 9(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.

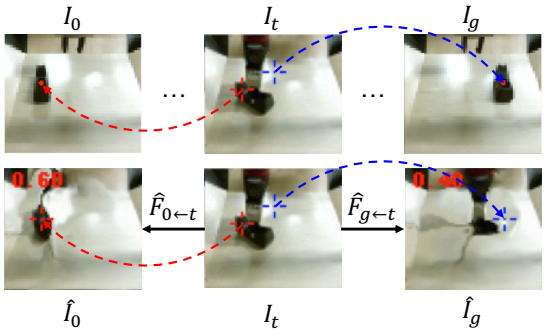


Fig. 8: Closed loop control is achieved by registering the current image I_t globally to the first frame I_0 and the goal image I_g. In this example registration to I_0 succeeds while registration to I_g fails since the object in I_g is too far away.

$$\hat{F}_{0\leftarrow t} = R(I_t, I_0) \qquad \hat{F}_{g\leftarrow t} = R(I_t, I_g) \tag{8}$$

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 8 for an illustration). There is no difference to the warping operation used in the video-prediction model, explained in section 4, equation 2:

$$\hat{I}_0 = \hat{F}_{0\leftarrow t} \diamond I_t \qquad \hat{I}_g = \hat{F}_{g\leftarrow t} \diamond I_t \tag{9}$$

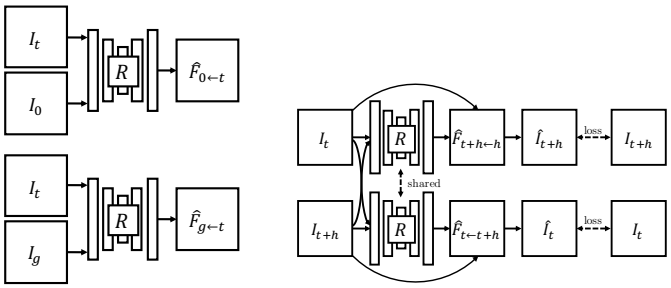
In essence for a current image $\hat{F}_{0\leftarrow t}$ puts I_t in correspondence with I_0, and $\hat{F}_{g\leftarrow t}$ puts I_t in correspondence with I_g. The motivation for registering to both I_0 and I_g is to increase accuracy and robustness. In principle, registering to either I_0 or I_g is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

$$\hat{d}_{0,t} = d_0 + \hat{F}_{0\leftarrow t}(d_0) \qquad \hat{d}_{g,t} = d_g + \hat{F}_{g\leftarrow t}(d_g) \tag{10}$$

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 10 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



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

Fig. 9: (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.

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:

$$\lambda_i = \frac{||I_i(d_i) - \hat{I}_i(d_i)||_2^{-1}}{\sum_j^N ||I_j(d_j) - \hat{I}_j(d_j)||_2^{-1}}. \tag{11}$$

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 **??**. 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.[2] Our approach is similar to the optic flow method proposed by [40]. 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.

$$\hat{I}_t = \hat{F}_{t\leftarrow t+h} \diamond I_{t+h} \qquad \hat{I}_{t+h} = \hat{F}_{t+h\leftarrow t} \diamond I_t \tag{12}$$

The network, which outputs $\hat{F}_{t\leftarrow t+h}$ and $\hat{F}_{t+h\leftarrow t}$, see Figure 9 (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 [40]. 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.

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. A classifier-based cost function allows the planner to discover any of the possible goal states.

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 goal images alone. Our approach is illustrated in Figure 11.

To train the few-shot classifier, we build upon model-agnostic meta-learning (MAML) [41], 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. [42] proposed an extension of MAML, referred to as concept acquisition through meta-learning (CAML), for learning new concepts from positive examples alone. We apply CAML to the setting of acquiring goal classifiers from positive examples.

Test Time Procedure At test time, the user provides a dataset $\mathcal{D}_j^+$ of K examples of successful end states for a new

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

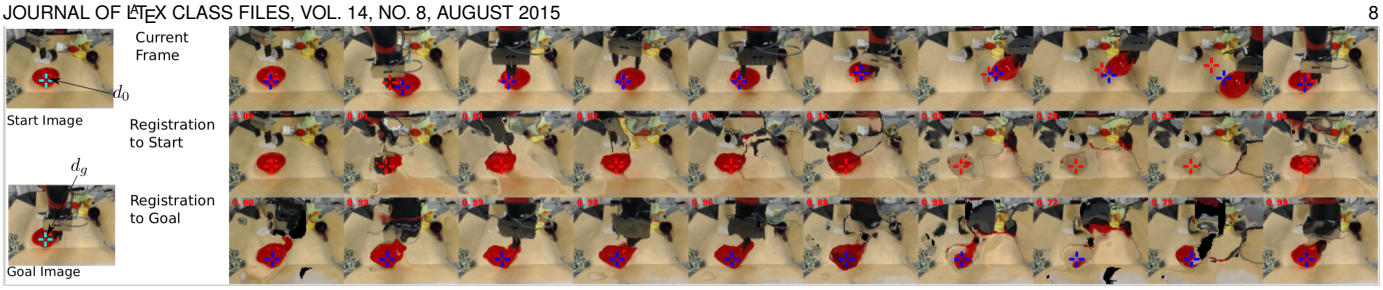


Fig. 10: Outputs of registration network. The first row shows the timesteps from left to right of a robot picking and moving a red bowl, the second row shows each image warped to the initial image via registration, and the third row shows the same for the goal image. A successful registration in this 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)

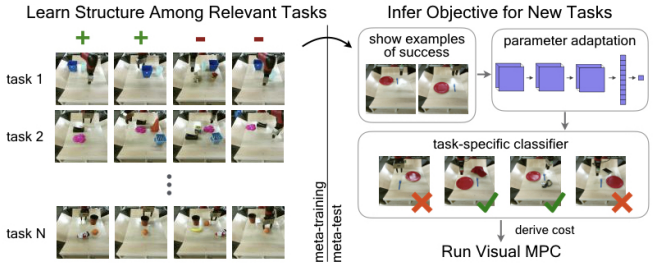


Fig. 11: We propose a framework for quickly specifying visual goals. Our goal classifier is meta-trained with positive and negative examples for diverse tasks (left), which allows it to meta-learn that some factors matter for goals (e.g., relative positions of objects), while some do not. At meta-test time, this classifier can learn goals for new tasks from a couple of 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.

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 [43] 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 classifier have the advantage that they allow moving objects precisely to target locations. Adding a registration mechanism allows for robust closed-loop control. The pixel-distance based cost function also has a high degree of robustness against distractor objects and clutter – an important feature when targeting diverse real-world environments.

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.

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.

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.

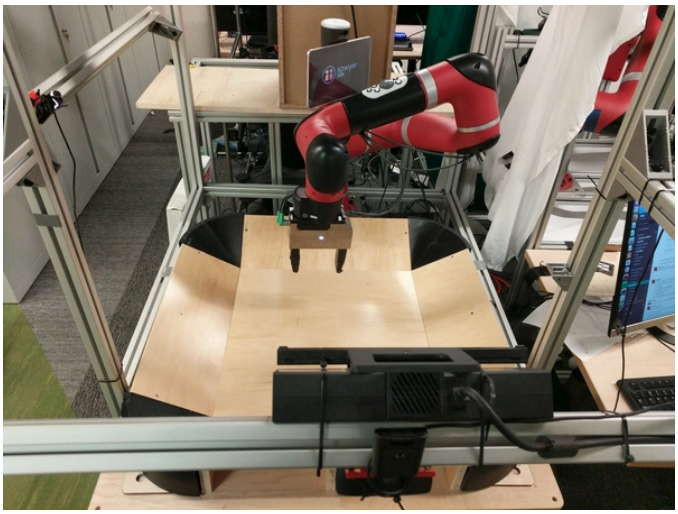


Fig. 12: Robot setup, with 2 standard web-cams arranged at different viewing angles.

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 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.

7 System Design

In this section we explain our robot setup and several critical system design choices and details about the data collection process.

Learning complex Pick and Place Behavior with Visual MPC: When collecting data by sampling from simple distributions, such as 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. We therefore explore how the sampling distribution used both for data collection and sampling-based planning can be changed to visit these, otherwise unlikely, states more frequently, allowing more complex behavior to emerge.

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 [45]. 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, see figure 16. 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.

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 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.

Multi-view visual MPC: Therefore we use a combination of multiple views, taken with multiple cameras arranged appropriately, to jointly define a 3D task. Figure 12 shows the robot setup, including 2 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 11). TO-DO: Figure XX shows an example lifting task, which specified in two views.

The classifier-based cost is also used in the multiview-setup, an example trajectory of cloth folding is shown TO-DO: in figure.

8 Experimental Evaluation

Our experimental evaluation consists of three parts, each answering one of the following three questions:

- How does a smooth cost function
- How does the skip-connection neural advection model compare to

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 was allowed to randomly move and pick up objects. The data collection process is fully autonomous, requiring human

	dist. ± std err. of mean	improvement ± std err. of mean
DNA with cost eqn. 7 [5]	24.6±2.35	2.1± 2.75
DNA with cost eqn. 6	**17.5 ± 2.37**	**8.3 ± 2.62**
SNA with cost eqn. 6 (ours)	18.18 ± 2.1	7.7 ± 2.33

TABLE 1: Results of the pushing benchmark on 20 different object/goal configurations. Units are pixels in the 64x64 images.

	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 2: 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.

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

TABLE 3: 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 $7.5cm$.

intervention only to replace and change out the objects in front of the robot.

The action space consisted of Cartesian movements along the x, y, and z axes, and for some parts of it we also added azimuthal rotations of the gripper. For evaluation, we selected novel objects that were never seen during training. The evaluation tasks required the robot to move objects in its environment from a starting state to a goal configuration, and performance was evaluated by measuring the distance between the final object position and goal position. TO-DO: how was it done for the classifier?

8.1 Evaluating Skip-connection Neural Advection

We first perform a quantitative comparison of visual-MPC using the proposed occlusion-aware SNA video prediction model and the expected distance cost with visual-MPC using the dynamic neural advection model (DNA) [5] with both the goal-point evaluation cost 7 and the expected distance cost 6.

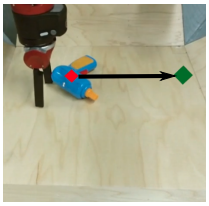


Fig. 13: Pushing task. The designated pixel (red diamond) needs to be pushed to the green circle.

We evaluate long pushes and multi-objective tasks where one object must be pushed without disturbing another. The supplementary video and links to the code and data are available at https://sites.google.com/view/sna-visual-mpc

Figure 13 shows an example task for the pushing benchmark. We collected 20 trajectories with 3 novel objects and 1 training object. Table 1 shows the results for the pushing benchmark. The column *distance* refers to the mean distance between the goal pixel and the designated pixel at the final time-step. The column *improvement* indicates how much the designated pixel of the objects was moved closer to their goal (or further away for negative values) compared to the starting location. The true locations of the designated pixels after pushing were annotated by a human labeler.

The results in Table 1 show that our proposed planning cost in Equation (6) substantially outperforms the planning cost used in prior work [5]. The performance of the SNA model in these experiments is comparable to the DNA model [5] when both use the expected-distance planning cost, since this task does not involve any occlusions.

To examine how well each approach can handle occlusions, we devised a second 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 goal object around it, as shown in Figure 14 on the left. While doing this, 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 picking one pixel on the target object, and one on the obstacle. The obstacle is commanded to remain stationary, by choosing the target position to be at the same location as the initial position. For the target object the destination is chosen on the other side of the obstacle.

We used four different object arrangements, with two training objects and two objects that were unseen during training. We found that, in most of the cases, the SNA model was able to find a valid trajectory, while the DNA model, that is not able to handle occlusion, was mostly unable to find a solution. Figure 14 shows an example of the SNA model successfully predicting the position of the obstacle through an occlusion and finding a trajectory that avoids the obstacle. These findings are reflected by our quantitative results shown in Table 2, indicating the importance of temporal skip connections.

8.2 Evaluating Closed-Loop Visual MPC

Videos and visualizations for closed-loop visual MPC can be found on this webpage: https://sites.google.com/view/robustness-via-retrying. We compare visual-MPC that uses a pixel-distance based cost based on our proposed self-supervised registration with visual-MPC using an off-the-shelf tracker, the "multiple instance learning tracker" MIL [46] from OpenCV. Note that all methods do not have any prior knowledge of objects – it is only provided with the position of one designed pixel in the initial and goal images, and must use the learned model to infer that this pixel belongs to an object that can be moved by the robot. Finally we also compare to visual MPC without registration method proposed by [6], which does not track the object explicitly, but relies on the flow-based video prediction model to

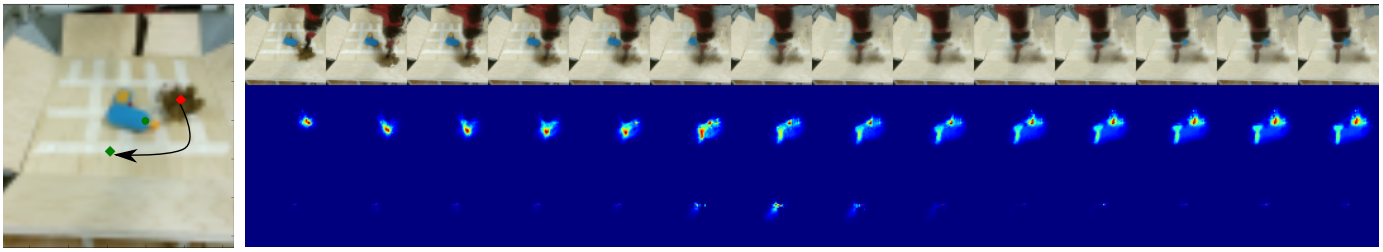


Fig. 14: Left: Task setup with green dot marking the obstacle. Right, first row: the predicted frames generated by SNA. Second row: the probability distribution of the designated pixel on the *moving* object (brown stuffed animal). Note that part of the distribution shifts down and left, which is the 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.

keep track of the designated pixel, which we call "predictor propagation."

8.2.1 Pushing with Retrying

For the first experiment, we disable the gripper control, which requires the robot to push objects to the target. We evaluate our method on 20 long-distance and 15 short-distance pushing tasks. For long-distance tasks the distance between the object and its goal-position is $30cm$ while for short-distance tasks it is $15cm$. Table 3 lists quantitative comparisons showing that on the long-distance benchmark visual-MPC using the proposed registrations approach not only outperforms prior work [6], but also outperforms the hand-designed, supervised object tracker [46]. By contrast for the short distance benchmark, all methods perform comparably. Thus theses results indicate the importance of closed loop control in long-horizon tasks. Using our learned registration, the robot is more frequently able to successfully recover after mispredictions or occlusions, an example is shown in Figure 15.

8.2.2 Combined Prehensile and Non-Prehensile Manipulation.

In the setting where the gripper is enabled it is part of the task to decide whether to solve a task by grasping or pushing the object to the goal. Similarly to the pushing setting we perform a benchmark where we define a set of 20 object relocation tasks and measure the final distance between the object and the target at the end of the episode. Interestingly we observe that in the majority of the cases the agent decides to grasp the object, as can be seen in the supplementary video.

8.3 Evaluating Classifier-based Cost Function

For evaluating the performance of the proposed classifier-based cost function, we study a visual object arrangement task, where different goals correspond to different relative arrangements of a pair of objects.

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 all approaches 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. Code for training the few-shot goal classifier and videos of our results are available at https://sites.google.com/view/few-shot-goals.

We qualitatively visualize the evaluation in Figure 18.TO-DO: explain how exactly you measure success 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 17, indicate that prior methods for learning distance metrics struggle to infer the goal of the task, while our approach leads to substantially more successful behavior on average.

8.4 Learning Cloth-Folding with Visual-MPC

a major impediment is that when executing random actions clothes can become tangled pu

9 Discussion

TO-DO: can cut from here: **Gradient-free vs. gradient based trajectory optimization** Prior work has also proposed to plan through learned models via differentiation, though not with visual inputs [47]. This can have the disadvantage of getting stuck in local minima. Also with gradient-based methods it is harder to ensure that the actions are within the distribution of actions sampled during data collection. This

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.

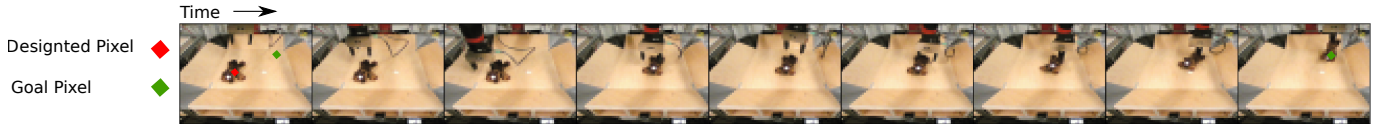


Fig. 16: Retrying behavior of our method combining prehensile and non-prehensile manipulation. In the first 4 time instants shown the robot pushes the object. It then loses the object, and decides to grasp it pulling it all the way to the goal. Retrying is enabled by applying the learned registration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

Fig. 17: Quantitative performance of visual planning for object rearrangement tasks across different goal specification methods: our meta-learned classifier, DSAE [30], 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.

is crucial to ensure that the model can make predictions based on a sufficient number of supporting data-points.

Instead using a model that predicts in pixel space, one could think of using a model making predictions in latent space. One potential advantage of latent space representations could be to handle *partially observed* settings, for example tasks where an object is occluded for extended periods of time.

10 Discussion and Future Work

We presented an algorithm that leverages supervision from visual prediction to learn a deep dynamics model, 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 model-predictive control.

The following has been identified as the most critical milestones for arriving at a general-purpose robotic manipulation system:

- **Replacing random data collection:** For learning how to screw a cap onto a bottle – a state that has exceptionally low probability of being reached by chance – data collection with random action is extremely ineffective. Therefore we suggest to collect data *on policy*, i.e. while executing Visual MPC targeting a specific goal. A promising way to define goals for data collection is by using human demonstrations, and since all attempts of reaching a goal, successful or not, contribute to the dataset, only small number of demonstrations would be needed.
- **Hierarchical Planning with Time-Agnostic Prediction:** Many tasks require reasoning over extended periods of time. A sampling based planning method would become prohibitively expensive for very long action sequences and also the prediction errors made by the model would accumulate, invalidating the plan. We therefore to add an additional long-horizon planning layer, that proposes subgoals which are within reach of visual MPC. We are currently exploring the use of a *Time-Agnostic Predictor* [48], a mechanism which instead of predicting states at fixed time-steps, predicts states at certain "bottleneck-state" which are easier to predict.
- **Policy Distillation and Action Proposals:** In high dimensional action spaces, with long planning horizons or non-smooth cost functions finding a trajectory that achieves low cost with CEM becomes increasingly harder. Furthermore limits on computational resources and timing requirements restrict the number of samples that can be taken in each planning step. One promising solution to this problem is learning the parameters of a distribution from which actions can be sampled more efficiently. Such an action proposal distribution could be learned for example by distilling the actions taken by visual MPC into a policy via supervised learning.

Acknowledgments

The authors would like to thank...

References

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

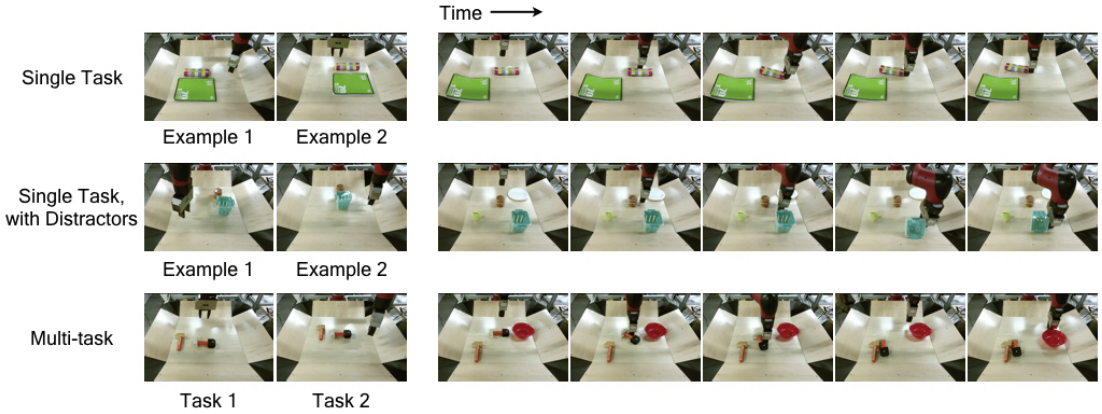


Fig. 18: Object arrangement performance of our goal classifier with distractor objects and with two tasks. The left shows a subset of the 5 positive examples that are provided for inferring the goal classifier(s), while the right shows the robot executing the specified task(s) via visual planning.

[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] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

[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] 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.

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

[27] 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.

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

[29] 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.

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

[31] 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.

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

[33] 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.

[34] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos

with scene dynamics," in *Advances In Neural Information Processing Systems*, 2016.
[35] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *International Conference on Learning Representations (ICLR)*, 2017.
[36] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool, "Dynamic filter networks," in *Neural Information Processing Systems (NIPS)*, 2016.
[37] 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.
[38] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," *arXiv preprint arXiv:1606.02378*, 2016.
[39] 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.
[40] S. Meister, J. Hur, and S. Roth, "Unflow: Unsupervised learning of optical flow with a bidirectional census loss," *arXiv preprint arXiv:1711.07837*, 2017.
[41] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *International Conference on Machine Learning (ICML)*, 2017.
[42] 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.
[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.
[44] D. P. Kroese, R. Y. Rubinstein, I. Cohen, S. Porotsky, and T. Taimre, "Cross-entropy method," in *Encyclopedia of Operations Research and Management Science*. Springer, 2013, pp. 326–333.
[45] D. Sherer, "Fetal grasping at 16 weeks' gestation," *Journal of ultrasound in medicine*, vol. 12, no. 6, 1993.
[46] 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.
[47] I. Lenz and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *In RSS*. Citeseer, 2015.
[48] D. Jayaraman, F. Ebert, A. A. Efros, and S. Levine, "Time-agnostic prediction: Predicting predictable video frames," *arXiv preprint arXiv:1808.07784*, 2018.
[49] 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.

APPENDIX A
SIMULATED EXPERIMENTS

A.0.1 Simulated Experiments

In order to provide a more controlled comparison, we also set up a realistic simulation environment using MuJoCo [49], 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.

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.