RL for Real: Reinforcement Learning with Deep Sensory Prediction Models for diverse Real-World Environments

TBD

Abstract—Applying deep reinforcement learning algorithms to robotic control tasks in complex, diverse, open-world environments without access to meticulously hand-engineered rewards still remains a challenge. We take a step towards solving this problem by drawing supervision from predicting future sensory observations. Concretely we propose to learn predictive models of future observations conditioned on the agent's future actions, and a method for planning actions under this model. We show that we can learn sensory prediction models effectively from real videos of a robot's experience, demonstrating that such models can handle diverse datasets with hundreds of objects, and that they enable the robot to accomplish a wide range of user-defined object manipulation goals, even when faced with novel objects. TO-DO: Page limit is 12!!

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

1 Introduction

Remarkably, humans are able to learn a range of complex yet generalizable behaviors when faced with a stream of high-dimensional sensory inputs and minimal external supervision.

The reinforcement learning (RL) problem statement covers such a real-world setting. Yet, the general setting of learning from raw sensory inputs from sparse rewards in the real world is incredibly challenging.

To make progress, many prior works have chosen various assumptions to make the problem more tractable. For instance, some works have been demonstrated only in settings where shaped rewards are available [**?**]. Others have assumed access to a perfect or hand-engineered simulator, from which millions of roll-outs can be taken [**?**], [**?**]. A few works have assumed that a low-dimensional, compact state representation is accessible during training [**?**], [1]. Finally, many works reset the state between roll-outs to a relatively narrow distribution of initial states, leading to a task is more repeatable [**?**], [**?**], [**?**], [2]. By making such assumptions, these prior methods have demonstrated the ability to learn complex visuomotor skills [1], learn Atari games from pixels [**?**], and master the game of Go [**?**]. However, to move these algorithms towards achieving general behaviors in the real world, it is arguably necessary to deploy them in diverse, real-world environments. Unfortunately, it is challenging to deploy such methods for learning in diverse, real-world environments, where shaped rewards, resets, compact state representations, and simulations are largely unavailable without extensive human involvement.

Unlike many prior works that focus on RL for mastery of highly-specialized skills in closed-world environments, here we instead focus on *generalization* in diverse environments. We consider a robotic control domain with a wide range of objects, where the algorithm has access to only raw sensor observations (i.e. pixels) and sparse rewards and has minimal control over the environment and how it can be reset. In such a setting with minimal assumptions, rewards are simply not enough supervision to be learned from alone. Instead, the supervision needs to come from the observations themselves. As such, we propose to learn action-conditioned predictive models directly on raw pixel observations. Beyond the necessity of dense supervision, sensory prediciton models have two additional benefits. They are goal-agnostic, which enables learning for a variety of different goals; and, by learning on top of raw observations, they are fully general in that they do not require a hand-engineered representation of the environment.

TO-DO: mention the autonomous data collection process

TO-DO: reference figure 1 here!

Despite the benefits, a number of challenges arise when aiming to use sensory prediction models: how can we learn a model of high-dimensional observations? how should an objective or reward be determined with respect to the predicted observations? how should experience be collected? how should actions be chosen with respect to the model? We discuss each of these design decisions and propose several options, weighting the benefits and trade-offs of each. Our overall approach amounts to a deep model-based reinforcement learning algorithm that leverages video prediction models to achieve a variety of pixel-based control tasks without shaped reward information.

The main contributions of this work are as follows. First, we propose a general framework for deep reinforcement learning with sensory prediciton models, abstracting away design decisions such as the model, the policy optimization, and the reward function. Second, we propose a model that can better maintain object permanence through occlusions via temporal skip connections. Third, we discuss several

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

Manuscript received xx.xx.xxx

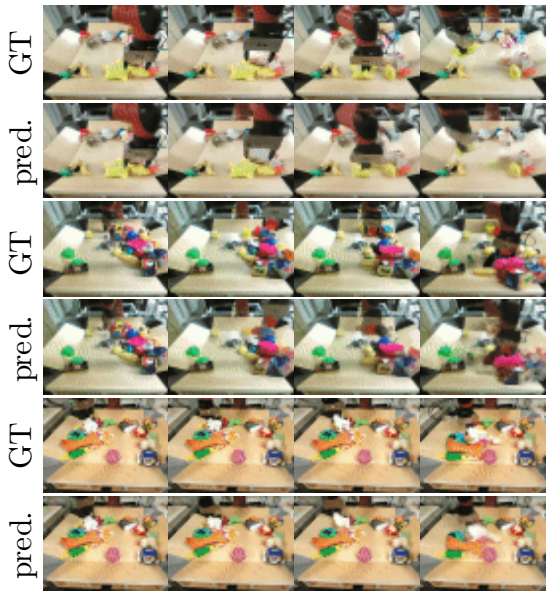


Fig. 1: TO-DO: Overview of visual MPC concept

forms of planning objectives, including goal pixel positions, goal image registration, and image classifiers. Finally, our evaluation demonstrates that these components can be combined to enable a real robot to perform a range of real-world object manipulation tasks from raw pixel observations. Our experiments include manipulation of previously unseen objects, handling multiple objects, pushing objects around obstructions, handling clutter, 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 paper builds upon prior conference papers [**?**]; our new contributions include an empirical evaluation of cloth manipulation and/or stochastic models, 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

2.1 Large-Scale, Self-Supervised Robotic Learning

In a number of recent works large-scale robotic data collection (often using multiple robots) has been the key factor enabling the automatic acquisition of generalizable features and skills. Several prior works have focused on autonomous data collection for individual skills, such as grasping [3], [4] or obstacle avoidance [5], [6]. In contrast to these methods, our approach learns predictive models that can be used to perform a variety of manipulations, and does not require a success measure or reward function during data collection.

In [7] Agrawal et al. propose learning an inverse model from raw sensory data without any supervision. While these methods demonstrated effective generalization to new objects, they were limited in the complexity of tasks and time-scale at which these tasks could be performed. The method was able to plan single pokes, and then greedily execute multiple pokes in sequence. 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 [8] 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.

2.2 Sensory Prediction Models

We propose to leverage sensory prediction models, in particular video-prediction, to achieve large-scale self-supervised learning. Relevant related work on video-prediction has been carried out in the context of synthetic video game images [9], [10] and robotic manipulation [11], [12], [13]. Video prediction without actions has been studied for unstructured videos [14], [15], [16] and driving [17], [18]. Several works have sought to use more complex distributions for future images, for example by using autoregressive models [13], [19]. 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 [12], [18]. Prior work has also sought to predict motion directly in 3D, using 3D point clouds obtained from a depth camera [20], 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.

3 Visual Model Predictive Control

In this section we define the visual model-predictive control problem formulation. We assume that the user defines a goal in terms of pixel motion by clicking on the object in the image and a corresponding target location, or by providing one or several demonstrations. Designated pixel positions and demonstrations can also be combined. Based on either of these task definitions we define per-time step cost functions c_t which are computed based on the results of the *action-conditioned* video-prediction model. To find an action sequence $a_{t_0:T}$ for which $c = \sum_{t=t_0}^{T} c_t$ over the time steps is minimal we use sampling based planning: A large number of candidate action sequences is sampled and the model's predictions are evaluated using c. The first action of the sequence which achieved lowest cost is applied to the robot.

To render the planning process more efficient we use the cross-entropy method (CEM), a gradient-free optimization procedure that consists of iteratively resampling action sequences and refitting Gaussian distributions to the actions with the best predicted cost. Further details can be found in section 6.

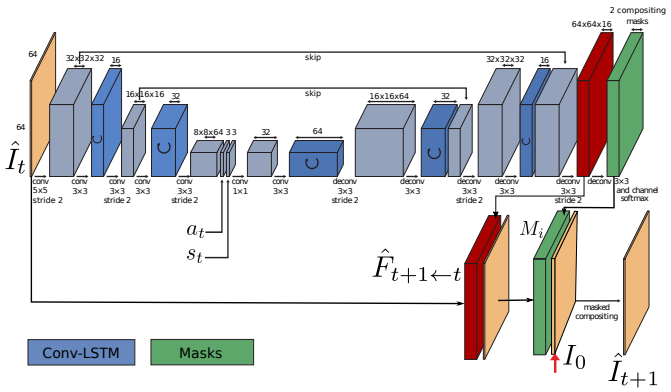


Fig. 2: Simplified SNA model based on Equation (4). The red arrow indicates where the image from the first time step I_0 is concatenated with the transformed images $\hat{F}_{t+1\leftarrow t} \diamond \hat{I}_t$ multiplying each channel with a separate mask to produce the predicted frame for step $t+1$.

To improve robustness against imperfect models, the actions are iteratively replanned at each real-world time step $\tau \in \{0, ..., \tau_{max}\}$ following the framework of model-predictive control (MPC): at each real-world step[1] τ, the model is used to plan T steps into the future, and the first action of the plan is executed.

4 Video Prediction for Control

When using a classifier based cost function as detailed in section 5.4 virtually any sensory prediction model can be used. However for the cost function based on user-defined start- and goal pixel positions it is necessary that the model is capable of predicting *where* certain points in the image are moving. The strengths and weaknesses of different costs functions and trade-offs between them are discussed in section 5.5.

4.1 Video Prediction via Pixel Transformations

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 selected start pixels $d_0^{(1)}, \ldots, d_0^{(P)}$ up to T steps into the future. To predict the trajectory of these pixels we leverage the model proposed in [12], where this flow prediction capability emerges implicitly, and therefore no external pixel motion supervision is required. Future images are generated by transforming past images. The model, shown in figure 2, uses stacked convolutional LSTMs that predict a 2 dimensional flow-field $\hat{F}_{t+1\leftarrow t}$ at each time step and uses it to transform a current image $\hat{I}_t$, similar to [21].

$$\hat{I}_{t+1} = \hat{F}_{t+1\leftarrow t} \diamond \hat{I}_t \tag{1}$$

where $\diamond$ denotes a bilinear interpolation operator that interpolates the pixel value bilinearly with respect to a location (x, y) and its four neighbouring pixels in the image.

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

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. To predict the future positions of the designated pixel[2] d, the same transformations which are used to transform the images are applied to the distribution over designated pixel location at time t for $P_{t,d}$ such that

$$\hat{P}_{t+1} = \hat{F}_{t+1\leftarrow t} \diamond \hat{P}_t \tag{2}$$

The distribution $\hat{P}_{t+1}$ is normalized at each time-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.

4.2 Skip Connection Neural Advection Model

To enable effective tracking of objects through occlusions, we propose an extension to the video-prediction model proposed in [12] that incorporates temporal skip connections.

To allow the model to handle occlusion during the prediction we now transform pixels not only from the previously generated image $\hat{I}_t$ but from all but from all previous images $\hat{I}_0, ...\hat{I}_t$ including the context image I_0 which is a real image.

$$\hat{I}_{t+1} = \mathbf{M}_0(\hat{F}_{t+1\leftarrow 0} \diamond I_t) + \sum_{j=1}^{\tau} \mathbf{M}_j(\hat{F}_{t+1\leftarrow j} \diamond \hat{I}_j) \tag{3}$$

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.

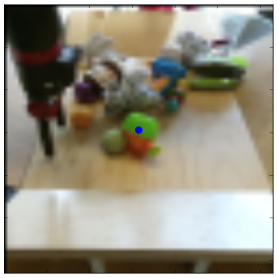


Fig. 3: The blue dot indicates the designated pixel

$$\hat{I}_{t+1} = \mathbf{M}_0 I_t + \mathbf{M}_1 \hat{F}_{t+1\leftarrow t} \hat{I}_t \tag{4}$$

2. For simplicity we assume that we only use a single designated pixel.

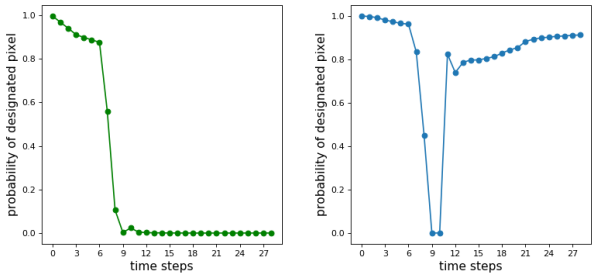


Fig. 4: Predicted probability $P_{d^{(0)}}(t)$ of the designated pixel being at the location of the blue dot indicated in Figure 3 for the DNA model (left) and the SNA model (right). TO-DO: can cut if no more space!

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 5. In this figure, the arm is predicted to move in front of the designated pixel, marked in blue in Figure 3. The predictions of the DNA model, shown in figure Figure 5(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 5(a).

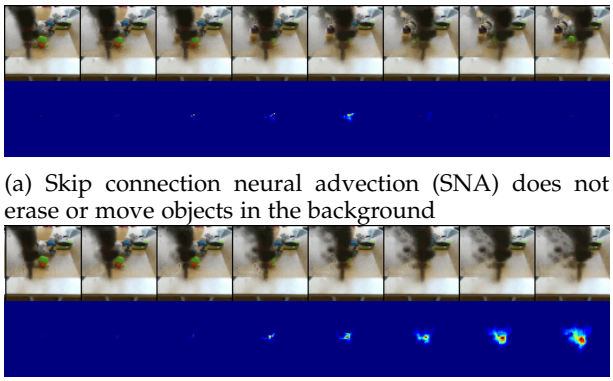
TO-DO: can cut from here: Next we show another illustration comparing the occlusion handling of DNA and the proposed SNA model. The graphs in Figure 4 show the predicted probability evaluated at the position of the designated pixel, shown in figure 3, which is stationary during the entire motion. Precisely 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 [12], the probability mass does not increase after the object reappears.

5 Planning Cost Functions

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.

Given a distribution over pixel positions $P_{t_0} \in \mathbb{R}^{H \times W}$ at time $t = 0$, the model predicts distributions over its



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

(b) Standard DNA [22] exhibits undesirable movement of the distribution $P_d(t)$ and erases the background

Fig. 5: Top rows: Predicted images of arm moving *in front of* green object with designated pixel (as indicated in Figure 3). Bottom rows: Predicted probability distributions $P_d(t)$ of designated pixel obtained by repeatedly applying transformations.

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 g, which is straight-forward to calculate from P_t and g.

$$c = \sum_{t=1,\ldots,T} c_t = \sum_{t=1,\ldots,T} \mathbb{E}_{\hat{d}_t \sim P_t} \left[\|\hat{d}_t - d_g\|_2 \right] \tag{5}$$

A different choice for the cost function is to evaluate P_t at the position of the goal-point, which describing how likely the model predicts an action sequence will hit the goal *exactly*, we call this method goal-point evaluation:

$$c = \sum_{t=1,\ldots,T} P_t(g) \tag{6}$$

The expected distance to the goal provides a smoother planning objective than the goal-point evaluation cost, and as our experiments show in section **??** enable 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. Also this cost 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 them according to a scheme discussed in Section 5.2.

5.2 Registration-based Cost

For 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 propose a method for registering the current image to both the start and goal image, where the designated pixels are known. Crucially, the registration method we propose 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.

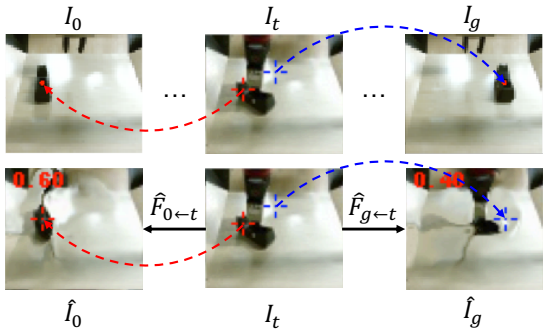
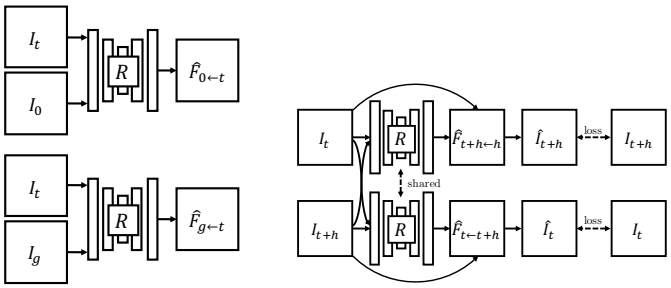


Fig. 6: Closed loop control is achieved by registering the current image I_t globally to the first frame I_0 and the goal image I_g. In this example registration to I_0 succeeds while registration to I_g fails since the object in I_g is too far away.



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

Fig. 7: (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.

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 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.

5.2.1 Test time procedure

We will first describe the registration scheme at test time (see Figure 7(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. Details about the archiecture are given in section 5.3. 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:

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

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 6 for an illustration):

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

where $\diamond$ denotes the same bilinear interpolation operator that is used in the video-prediction model, see equation 1. 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 8 visualizes an example tracking result while the gripper is moving an object.

5.2.2 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

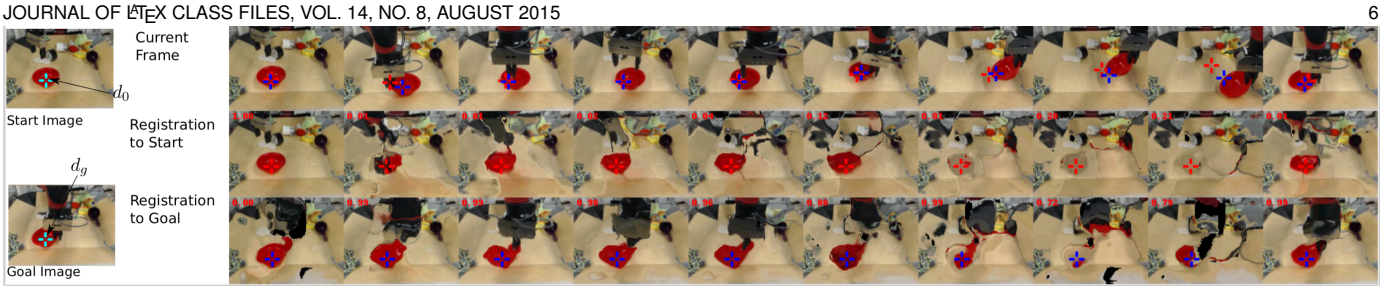


Fig. 8: Outputs of registration network. The first row shows the timesteps from left to right of a robot picking and moving a red bowl, the second row shows each image warped to the initial image via registration, and the third row shows the same for the goal image. A successful registration in this 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)

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 5) 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 7. The index i then also loops over the views and indices of the designated pixels.

5.3 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 [23]. 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 7 (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 [23]. 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

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.

5.4 Classifier-based Cost

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 $\mathcal{T}_j$.

5.4.1 *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.

To train the few-shot classifier, we build upon model-agnostic meta-learning (MAML) [24], 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. [25] 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.

5.4.2 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. Concretely, the cost is defined as the negative of the classifier prediction.

5.4.3 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 [26] 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. TO-DO: add images for classifier!

5.5 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

Prior work has also proposed to plan through learned models via differentiation, though not with visual inputs [27]. We instead use a stochastic, sampling-based planning method [22], [28], which we extend to handle a mixture of continuous and discrete actions

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.

In the visual MPC setting the action sequences found by the optimizer can be very different between execution times steps τ (not to be confused with prediction time steps t). 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. Naive 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 show that 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 prior work [29] 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 time step as the mean. 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 Scaling up Visual Model-Predictive Control

To allow disambiguating 3D-tasks, we extend visual MPC to include multiple camera views. Since tasks are defined in terms of pixel motion in 2D image space, the combination of multiple 2D tasks defines a 3D task, when cameras are oriented appropriately. In our experiments, we show that we can define 3D manipulation tasks, such as lifting an object from the table, that would be ambiguous using only a single camera view. 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).

In prior work on video-prediction-based robotic manipulation [22], [29], the capabilities that emerged out of self-supervised learning were generally restricted to pushing and dragging objects. To enable more complex tasks, we also explore how visual MPC can enable behaviors that include picking and lifting objects for rearrangement. One of the main challenges with this is that random exploration is unlikely to pick up objects a sufficiently large fraction of the time to allow the model to learn grasping skills. To

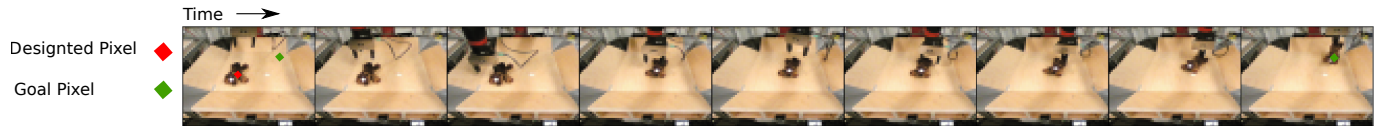


Fig. 9: Retrying behavior of our method combining prehensile and non-prehensile manipulation. In the first 4 time instants shown the agent pushes the object. It then loses the object, and decides to grasp it pulling it all the way to the goal. Retrying is enabled by applying the learned registration to both camera views (here we only show the front view).

alleviate this challenge, 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 [30]. 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 9. 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.

8 Experimental Evaluation

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 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) [22] with both the goal-point evaluation cost 6 and the expected distance cost 5.

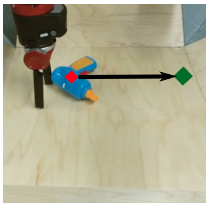


Fig. 10: Pushing task. The designated pixel (red diamond) needs to be pushed to the green circle.

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

	dist. mean, std.	improvement mean, std.
DNA with cost eqn. 6 [22]	24.6±10.6	2.1±12.4
DNA with cost eqn. 5	**17.5 ± 10.2**	**8.3 ± 11.8**
SNA with cost eqn. 5 (ours)	18.18 ± 9.5	7.7 ± 10.5

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

	moved imp. mean, std.	stationary imp. mean, std.
DNA [22]	0.83 ±1.13	-1.1 ± 0.9
SNA	**10.6 ± 3.7**	**-1.5 ± 0.9**

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.

Figure 10 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.

The results in Table 1 show that our proposed planning cost in Equation (5) substantially outperforms the planning cost used in prior work [22]. The performance of the SNA model in these experiments is comparable to the DNA model [22] 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 11 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.

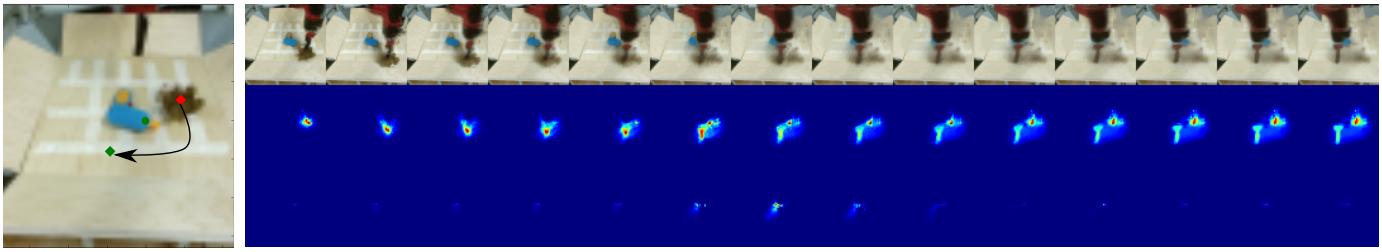


Fig. 11: Left: Task setup with green dot marking the obstacle. Right, first row: the predicted frames generated by SNA. Second row: the probability distribution of the designated pixel on the *moving* object (brown stuffed animal). Note that part of the distribution shifts down and left, which is the 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.

	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, where the complete image has size 48x64.

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 11 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 [31] 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 [29], which does not track the object explicitly, but relies on the flow-based video prediction model to 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 [29], but also outperforms the hand-designed, supervised object tracker [31]. 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 12.

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. We evaluate our learned classifier on the predictions made by the video prediction model TO-DO: unclear, how exactly do you evaluate the classifier? and derive the cost used for planning from the predicted probability of success.

8.3.1 Real-World Experiments

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

Fig. 12: 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.

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.

We qualitatively visualize the evaluation in Figure **??**.TODO: 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 **??**, 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.

9 DISCUSSION

10 CONCLUSION

The conclusion goes here.

ACKNOWLEDGMENTS

The authors would like to thank...

REFERENCES

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end learning of deep visuomotor policies," *Journal of Machine Learning Research (JMLR)*, 2016.
[2] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016.
[3] 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.
[4] 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.
[5] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.
[6] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," *arXiv preprint arXiv:1704.05588*, 2017.
[7] 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.
[8] T. Kurutach, A. Tamar, G. Yang, S. Russell, and P. Abbeel, "Learning plannable representations with causal infogan," *arXiv preprint arXiv:1807.09341*, 2018.
[9] 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.
[10] S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed, "Recurrent environment simulators," *arXiv preprint arXiv:1704.02254*, 2017.
[11] 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.
[12] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Neural Information Processing Systems (NIPS)*, 2016.
[13] 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.
[14] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," *International Conference on Learning Representations (ICLR)*, 2016.
[15] 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.
[16] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *Advances In Neural Information Processing Systems*, 2016.
[17] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *International Conference on Learning Representations (ICLR)*, 2017.
[18] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool, "Dynamic filter networks," in *Neural Information Processing Systems (NIPS)*, 2016.
[19] 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.
[20] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," *arXiv preprint arXiv:1606.02378*, 2016.
[21] 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.
[22] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *International Conference on Robotics and Automation (ICRA)*, 2017.
[23] S. Meister, J. Hur, and S. Roth, "Unflow: Unsupervised learning of optical flow with a bidirectional census loss," *arXiv preprint arXiv:1711.07837*, 2017.
[24] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *International Conference on Machine Learning (ICML)*, 2017.
[25] 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.
[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.
[27] I. Lenz and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *In RSS*. Citeseer, 2015.

[28] R. Y. Rubinstein and D. P. Kroese, *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.
[29] 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
[30] D. Sherer, "Fetal grasping at 16 weeks' gestation," *Journal of ultrasound in medicine*, vol. 12, no. 6, 1993.
[31] 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.
[32] 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.

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.

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 [32], 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 [29]), 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.