Visual Robot Task Planning

Chris Paxton, Yotam Barnoy, Kapil Katyal, Raman Arora and Gregory D. Hager
Department of Computer Science
The Johns Hopkins University, Baltimore, Maryland 21218
Email: {cpaxton, ybarnoy1, kkatyal2, rarora8, ghager1}@jhu.edu

***Abstract*—Prospection, the act of predicting the consequences of many possible futures, is intrinsic to human planning and action, and may even be at the root of consciousness. Surprisingly, this idea has been explored comparatively little in robotics. In this work, we propose a neural network architecture and associated planning algorithm that (1) learns a representation of the world useful for generating prospective futures after the application of high-level actions from a large pool of expert demonstrations, (2) uses this generative model to simulate the result of sequences of high-level actions in a variety of environments, and (3) uses this same representation to evaluate these actions and perform tree search to find a sequence of high-level actions in a new environment. Models are trained via imitation learning on a variety of domains, including navigation, pick-and-place, and a surgical robotics task. Our approach allows us to visualize intermediate motion goals and learn to plan complex activity from visual information.**

I. Introduction

Humans are masters at solving problems they have never encountered before. When attempting to solve a difficult problem, we are able to build a good abstract models and to picture what effects our actions will have. Some say this act — the act of prospection — is the essence of true intelligence [19]. If we want robots that can plan and act in general purpose situations just as humans do, this ability would appear crucial.

As an example, consider the task of stacking a series of colored blocks in a particular pattern, as explored in prior work [27]. A traditional planner would view this as a sequence of high-level actions, such as `pickup(block)`, `place(block,on_block)`, and so on. The planner will then decide which object gets picked up and in which order. Such tasks are often described using a formal language such as the Planning Domain Description Language (PDDL) [8]. To execute such a task on a robot, specific goal conditions and cost functions must be defined, and the preconditions and effects of each action must be specified. This is a time consuming manual undertaking [2]. Humans, on the other hand, do not require that all of this information to be given to them beforehand. We can learn models of task structure purely from observation or demonstration. We work directly with high dimensional data gathered by our senses, such as images and haptic feedback, and can reason over complex paths without being given an explicit model or structure.

Ideally, we would learn representations that could be used for all aspects of the planning problem, that also happen to be human-interpretable. A recent line of work in robotics focuses on making structured predictions to inform planning [6, 5, 26, 16]: So far, however, these approaches focus on making relatively short-term predictions, and do not take into account high-level variation in how a task can be performed. One-shot deep imitation learning can produce general-purpose models for various tasks, but relies on a task solution from a human expert, and does not generate prospective future plans that can be evaluated for reliable performance in new environments [27]; these approaches are also very data intensive.

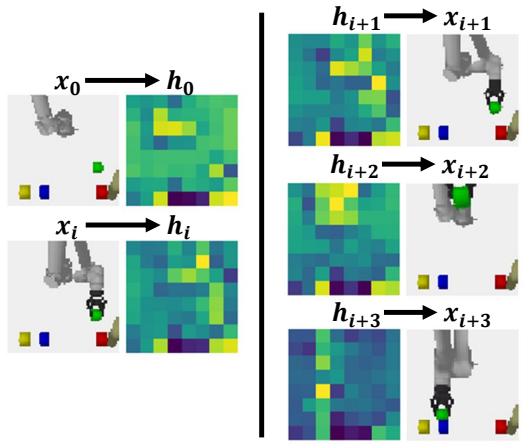

Figure 1: Example of our algorithm using learned policies to predict a good sequence of actions. Left: initial observation x_0 and current observation x_i, plus corresponding encodings h_0 and h_i. Right: predicted results of three sequential high level actions.

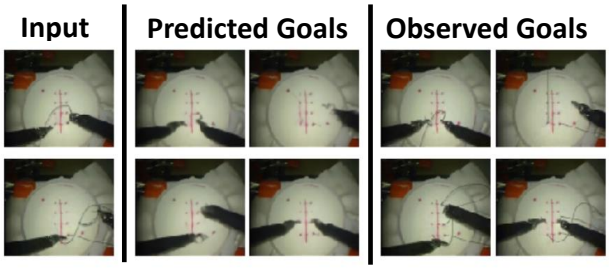

Figure 2: Predicting the next step during a suturing task based on labeled surgical data. Predictions clearly show the next position of the arms.

We propose a supervised model that learns high-level task structure from imperfect demonstrations. Our approach then generates interpretable task plans by predicting and evaluating a sequence of high-level actions, as shown in Fig. 1. In Fig. 1, we predict the action that is most likely to succeed from the observed state of the world, imagine what the result of that action will be, and repeat.

Models are learned from labeled training data containing both successes and failures, are not reliant on a large number

of good expert examples, and work in a number of domains including navigation, pick-and-place, and robotic suturing (Fig. 2). We also describe a planning algorithm that simulates a set of possible futures and choose the best sequence of high-level actions to execute, resulting in realistic explorations of possible futures.

To summarize, our contributions are:[1]

- A network architecture and training methodology for learning a deep representation of a planning task.
- An algorithm to employ this planning task to generate and evaluate sequences of high-level actions.
- Experiments demonstrating the model architecture and algorithm on multiple datasets.

II. Related Work

Motion Planning: In robotics, effective TAMP approaches have been developed for solving complex problems involving spatial reasoning [22]. A subset of planners focused on Partially Observed Markov Decision Process extend this capability into uncertain worlds; examples include DeSPOT, which allows manipulation of objects in cluttered and challenging scenes [14]. These methods rely on a large amount of built-in knowledge about the world, however, including object dynamics and grasp locations.

A growing number of works have explored the integration of planning and deep neural networks. For example, QMDP-nets embed learning into a planner using a combination of a filter network and a value function approximator network [11]. Similarly, value iteration networks embed a differentiable version of a planning algorithm (value iteration) into a neural network, which can then learn navigation tasks [21]. Vezhnevets proposed to generate plans as sequences of actions [25]. Other prior work employed Monte Carlo Tree Search (MCTS) together with a set of learned action and control policies for task and motion planning [17], but did not incorporate predictions.

Prediction is intrinsic to planning in a complex world. While most robotic motion planners assume a simple causal model of the world, recent work has examined learning predictive models. Lotter et al. [15] propose PredNet as a way of predicting sequences of images from sensor data, with the goal of predicting the future, and Finn et al. [5] use unsupervised learning of predictive visual models to push objects around in a plane. However, to the best of our knowledge, ours is the first work to use prospection for task planning.

Learning Generative Models.: GANs are widely considered the state of the art in learned image generation [3, 1], though they are far from the only option. The Wasserstein GAN is of particular note as an improvement over other GAN architectures [1]. Isola et al. proposed the PatchGAN, which uses an average adversarial loss over "patches" of the image, together with an L1 loss on the images as a way of training conditional GANs to produce one image from another [10]. Prior work has examined several ways of generating multiple realistic predictions [18, 3, 7] More recently, [7] proposed to learn a deep predictive network that uses a stochastic policy over goals for manipulation tasks, but without the goal of additionally predicting the future world state.

Learning Representations for Planning: Sung et al. [20] learn a deep multimodal embedding for a variety of tasks. Finn et al. [5] learn a deep autoencoder as a set of convolutional blocks followed by a spatial softmax; they found that this representation was useful for reinforcement learning. Recently, Higgins et al. [9] proposed DARLA, the DisentAngled Representation Learning Agent, which learns useful representations for tasks that enable generalization to new environments. Interpretability and predicting far into the future were not goals of these approaches.

Finally, we must note current work in learning model-based planning from scratch. Weber et al. [26] propose imagination-based agents for reinforcement learning, but these include very simple tasks with discrete actions. Other work has looked at deep models for model predictive control [12, 6] or model-based RL [16]. These operate over shorter horizons than our work, and are not attempting to capture high-level task structure in their predictions. To our knowledge, ours is the first work examining deep generative models for making high-level predictions about the world for task planning.

III. Approach

We define a planning problem with continuous states $x \in \mathcal{X}$, where x contains observed information about the world such as a camera image, and a set of high-level actions $a \in A$ that describe the task in semantically meaningful terms (e.g. "grab the red block") and can be encoded as integers.

Our objective is to learn a set of models representing the necessary components of this planning problem, but acting in this latent space $\mathcal{H}$. In other words, given a particular action a and an observed state x, we want to be able to predict both an end state x' and the optimal sequence of actions $a \in A^*$ necessary to take us there. We specifically propose that there are three components of this prediction function:

1. $f_{enc}(x) \rightarrow h \in \mathcal{H}$, an encoder mapping observations to hidden states.
2. $f_{dec}(h) \rightarrow (x)$, a decoder mapping from hidden state to expected observations.
3. $T(h, a) \rightarrow h' \in \mathcal{H}$, maps between different hidden states given a high-level action a.

In practice, we include the hidden state of the first world observation as well in our transform function, in order to capture any information about the world that may be occluded. This gives the transform function the form:

$$T(h_0, h, a) \rightarrow h' \in \mathcal{H}$$

We assume the hidden state h contains all the necessary information about the world to make high level decisions as long as this h_0 is available to capture change over time. As such, we can learn additional functions representing the value of a given hidden state, the predicted value of actions moving forward from each hidden state, and connectivity between

[1]Code and data is available at https://cpaxton.github.io/costar_plan/

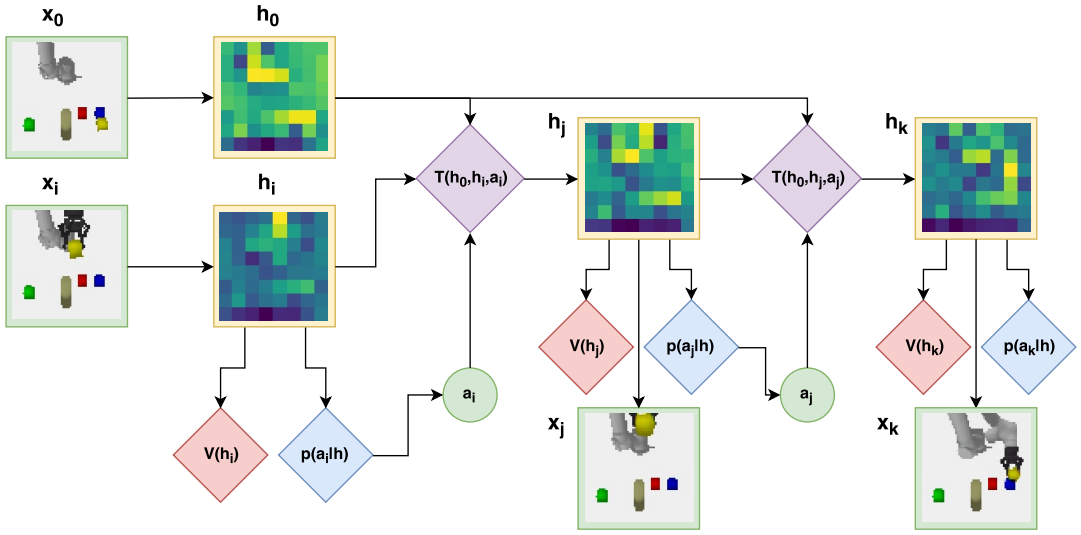

Figure 3: Overview of the prediction network for visual task planning. We learn $f_{enc}(x)$, $f_{dec}(x)$, and $T(h, a)$ to be able to predict and visualize results of high-level actions. We map from observations $x \in \mathcal{X}$ to hidden states $h \in \mathcal{H}$, and then use a learned permissability function p to determine which actions $a \in \mathcal{A}$ can be taken. The results of these actions h' are predicted with $h' = T(h_0, h, a)$.

hidden states. Given these components, we can appropriately represent the task as a tree search problem.

A. Model Architecture

Fig. 3 shows the architecture for visual task planning. Inputs are two images x_0 and x_i: the initial frame when the planning problem was first posed, and the current frame. We include x_0 and h_0 to capture occluded objects and changes over time from the beginning of the task. In Fig. 3, hidden states h_0, h_i, h_j, h_k are represented by averaging across channels.

Encoder and Decoder. We train f_{enc} and f_{dec} using the encoder-decoder architecture shown in Fig. 4 to find the mapping in and out of $\mathcal{H}$. Convolutional blocks are indicated with Ck, where k is the number of filters. Most of our layers are 5×5 convolutions, although we used a 7×7 convolution on the first layer and we use 1×1 convolutions to project into and out of the hidden space. Each convolution is followed by an instance normalization and a ReLU activation. Stride 2 convolutions and transpose convolutions are then used to increase or decrease image size after each block. The final projection into the hidden state has a sigmoid activation in place of ReLU and is not paired with a normalization layer. In most of our examples, this hidden space is scaled down to an $8 \times 8 \times 8$ space.

Both dropout and normalization played an important role in learning fast, accurate models for encoding the hidden state, but we use the instance norm instead of the more common batch normalization in order to avoid issues with dropout. After every block, we add a dropout layer D, with dropout initially set to 10%. Instance normalization has been found useful for image generation in the past [23]. We do not apply dropout at test time except with GANs.

Transform function. $T(h_0, h, a)$ computes the most likely next hidden state. This function was designed to combine information about the action and two observed states, and to compute global information over the entire hidden space. We use the spatial soft argmax previously employed by Finn and Levine. [13, 6] and Ghadirzadeh et al. [7] to compute a set of keypoints, which we then concatenate with a high-level action label and use to predict a new image. This is sufficient to capture the next action with a good deal of fidelity (see Sec. V-B), but to capture background details we add a skip connection across this spatial soft argmax bottleneck. In practice, given $f_{enc}(x)$ and $f_{dec}(x)$, we train $T(h_0, h, a)$ as an action subgoal prediction function, which is a mapping $f_{dec}(T(f_{enc}(x_0), f_{enc}(x), a)) \rightarrow (x')$.

Fig. 5 shows the complete transform block as used in the block stacking and navigation case studies described below. For the suturing case study, with a larger input and hidden space, we add an extra set of size 64 convolutions to each side of the architecture and a corresponding skip connection, but it is otherwise the same. Each dense or convolutional layer is followed by a ReLU nonlinearity. The final projection into the hidden state also has a sigmoid activation and no instance normalization layer, as in the encoder.

Value functions. $V(h)$ computes the value of a particular hidden state h as the probability the task will be successful from that point onwards, and $Q(h_0, h, a, a')$ predicts the probability that taking action a' from the tree search node (h_0, h, a) will be successful. These are trained based on $\{0, 1\}$ labels indicating observed task success and observed failures. We also train the function $f(h_0, h, a)$ which predicts whether or not an action a successfully finished.

Structure prior. Value functions do not necessarily indicate what happens if there are no feasible actions from a particular state. To handle this, we learn the permissability function $p(a'|h_0, h, a)$, which states that it is possible for a' to follow a, but does not state whether or not a' will succeed.

These last four models are trained on supervised data, but without the instance normalization in f_{emc}, f_{dec}, and T, as we saw this hurt performance. Q, p, and f were trained with

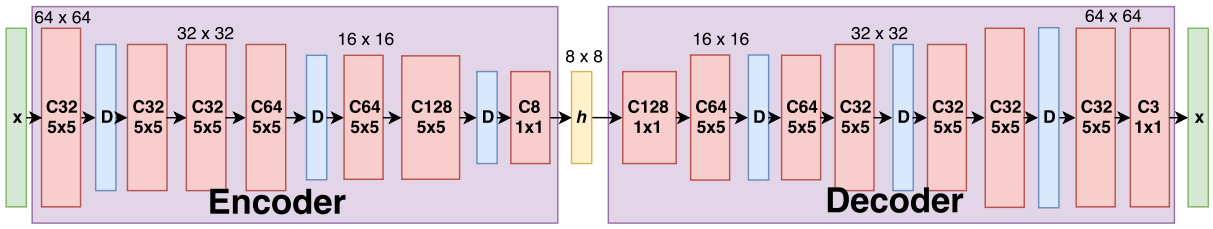

Figure 4: Encoder-decoder architecture used for learning a transform into and out of the hidden space h.

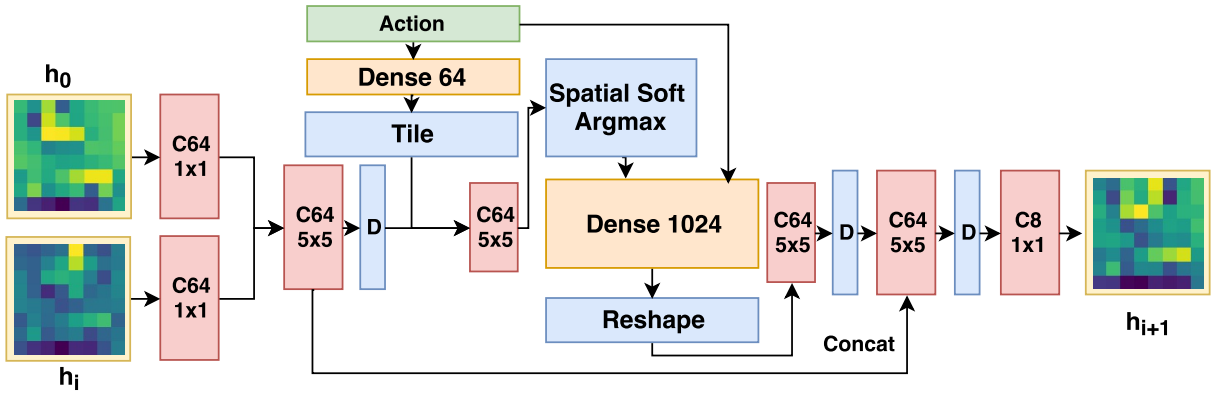

Figure 5: Architecture of the transform function $T(h_0, h, a)$ for computing transformations to an action subgoal in the learned hidden space.

two 1×1 convolutions on h and h_0, then a concatenated, followed by $C64 - C64 - FC256 - FC128$, where FCk is a fully connected layer with k neurons. The value function $V(h)$ was a convolutional neural net of the form $C32 - C64 - C128 - FC128$.

B. Learning

We train our predictor directly on supervised pairs containing the state $x' = f_{dec}(T(f_{enc}(x), a))$ resulting from action a. First, we considered a simple L1 loss on the output images. However, this might not capture all details of complex scenes, so we also train with an augmented loss which encourages correct classification of the resulting image. This approach is in some ways similar to that used by [3], in which the authors predict images while minimizing distance in a feature space trained on a classification problem. Here, we use a combination of an L1 term and a term maximizing the cross-entropy loss on classification of the given image, which we refer to as the L1+λC loss in the following, where λ is some weight.

This classifier predicts, given an observation of a subgoal x_g, which action was performed to take us to this point. This classification function should capture salient features of a particular high-level action. We also explored using two different GAN losses: the Wasserstein GAN [1] and the pix2pix GAN from Isola et al. [10].

First we train the goal classifier $C(x)$ on labeled training data for use in testing and in training our augmented loss. Next we train $f_{enc}(x)$ and $f_{dec}(x)$, which provide our mapping to and from the hidden world state. Finally we train the transform function $T(h_0, h, a)$ and the evaluation functions used in our planning algorithm.

Transform Training. To encourage the model to make predictions that remain consistent over time, we link two consecutive transforms with shared weights, and train on the sum of the L1 loss from both images, with the optional classifier loss term applied to the second image. The full training loss given ground truth predictions $\hat{x}_1$, $\hat{x}_2$ is then:

$$\mathcal{L}(x_1, x_2) = \|\hat{x}_1 - x_1\|_1 + \|\hat{x}_2 - x_2\|_1 + \lambda C(x_2)$$

Implementation. All models were implemented in Keras and trained using the Adam optimizer, except for the Wasserstein GAN, which we trained with RMSProp as per prior work [1]. We performed multiple experiments to set the learning and dropout rates in our models, and selected a relatively low learning rate of $1e-4$ and dropout rate of 0.1, which strikes a balance between regularization and crisp per-pixel predictions when learning the hidden state.

C. Visual Planning with Learned Representations

We use these models together with Monte Carlo Tree Search (MCTS) in order to find a sequence of actions that we believe will be successful in the new environment. The general idea is that we run a loop where we repeatedly sample a possible action a' according to the learned function $Q(h_0, h, a, a')$ and use this action to simulate the effects of that high-level action $T(h_0, h, a')$. We can then execute the sequence of learned or provided black box policies to complete the motion on the robot.

We propose a variant of MCTS as a general way of exploring the tree over possible actions [17]. We represent each

Algorithm 1 Algorithm for visual task planning with a learned state representation.

Given: max depth d, initial state x_0, current state x, number of samples $N_{samples}$
$h = f_{enc}(x)$, $h_0 = h$
for $i \in N_{samples}$ **do**
  EXPLORE(h_0,h,$\emptyset$,0,d)
end for
function EXPLORE(h_0,h,a,i,d)
  v_i = EVALUATE(h_0, h, a)
  if $i \geq d$ or $v_i < v_{failed}$ **then return** v_i **end if**
  a' = SAMPLE(h_0, h, a)
  $h' = T(h_0, h, a')$
  v' = EXPLORE(h_0,h',a',$i+1$,d)
  UPDATE(a, a', v')
  return $v_i \cdot v'$
end function

node in the tree by a unique instance of a high-level action ($\emptyset$ for the root). The full algorithm is described in Alg. 1.

The EVALUATE function sets $v_i = V(h)$, but also checks the validity of the chosen action and determines which actions can be sampled. We also compute $f(h_0, h, a)$ to determine if the robot would succesfuuly complete the action with some confidence c_{done}. If not this is considered a failure ($v_i = 0$). If $v' < v_{failed}$, we will halt exploration.

The SAMPLE function greedily chooses the next action a' to pursue according to a score $v(a, a')$:

$$v(a, a') = \frac{cQ(h_0, h, a, a')}{N(a, a')} + v^*(a, a')$$

where Q is the learned action-value function, $N(a, a')$ is the number of times a' was visited from a, and $v^*(a, a')$ is the best observed result from taking a'. We set $c = 10$ to encourage exploration to actions that are expected to be good. The UPDATE function is responsible for incrementing $N(a, a')$. Sampled actions a' are rejected if we predict a' is not reachable from its parent.

IV. EXPERIMENTAL SETUP

We applied the proposed method to both a simple navigation task using a simulated Husky robot, and to a UR5 block-stacking task. [2]

In all examples, we follow a simple process for collecting data. First, we generate a random world configuration, determining where objects will be placed in a given scene. The robot is initialized in a random pose as well. We automatically build a task model that defines a set of control laws and termination conditions, which are used to generate the robot motions in the set of training data. Legal paths through this task model include any which meet the high-level task specification, but may still violate constraints (due to collisions or errors caused by stochastic execution).

We include both positive and negative examples in our training data. Training was performed with Keras [4] and Tensorflow 1.5 for 45,000 iterations on an NVIDIA Titan Xp GPU, with a batch size of 64. Training took roughly 200 ms per batch.

[2]Source code for all examples will be made available after publication.

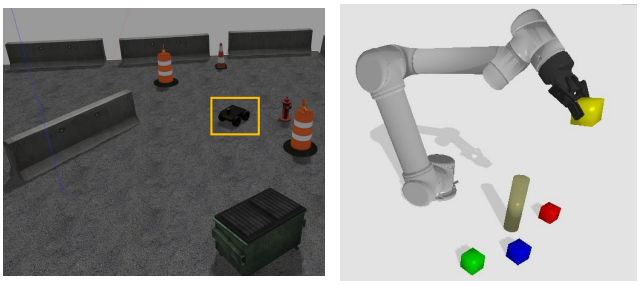

Figure 6: Simulation experiments. Left: Husky navigation task. The robot is highlighted. Right: UR5 block-stacking task with obstacle avoidance.

A. Robot Navigation

In the navigation task, we modeled a Husky robot moving through a construction site environment to investigate one of four objects: a barrel, a barricade, a construction pylon or a block, as shown in Fig. 6(left). The goal was to find a path between different objects, so that it takes less then ten seconds to move between any two objects. Here, x was a 64x64 RGB image that provides an aerial view of the environment. Data was collected using a Gazebo simulation of the robot navigating to a randomly-chosen sequence of objects. We collected 208 trials, of which 128 were failures.

B. Simulated Block Stacking

To analyze our ability to predict goals for task planning, we learn in a more elaborate environment. In the block stacking task, the robot needed to pick up a colored block and place it on top of any other colored block. The robot succeeds if it manages to stack any two blocks on top of one another and failed immediately if either it touches this obstacle or if at the end of 30 seconds the task has not been achieved. Training was performed on a relatively small number of examples: we used 6020 trials, of which 2991 were successful and 3029 were failures. The state x is a 64×64 RGB image of the scene from a fixed external camera.

We provided a set of non-optimal expert policies and randomly sampled a set of actions. This task was fairly complicated, with a total of 36 possible actions divided between two sub-tasks. Separate high-level actions were provided for aligning the gripper with an object, moving towards a grasp, closing the gripper, lifting an object, aligning a block with another block below it, stacking the currently held block on another, opening the gripper, and returning to the home position, for each of the four blocks in the scene.

Each performance was labeled a failure if either (a) it took more than 30 seconds, (b) there was a collision with the obstacle, or (c) the robot moved out of the workspace for any reason. The simulation was implemented in PyBullet, and execution was stochastic and unpredictable. At times the robot would drop the currently-held block, or it would fail to accurately place the block held in its hands. There was also some noise in the simulated images.

C. Surgical Robot Image Prediction

Next, we explored our ability to predict the goal of the next motion on a real-world surgical robot problem. Minimally invasive surgery is a highly skilled task that requires a great deal of training; our image prediction approach could allow novice users some insight into what an expert might do in their situation. There is a growing amount of surgical robot video available, and a growing body of work seeks to capitalize on this to improve video prediction [24]. We used a subset of the JIGSAWS dataset to train a variant of our Visual Task Planning models on a labeled suturing task in order to predict the results of certain motions.

The JIGSAWS dataset consists of stereo video frames, with each pair labeled as belonging to one of 15 possible gestures. For our task, We used only the left frames of the video stream. The dataset contains the 3 tasks of suturing, know tying, and needle passing, with each task consisting of a subset of gestures. We reduced the image dimensions from 640×480 to a more reasonable 96×128. For this application, we used a slightly larger $12 \times 16 \times 8$ hidden representation.

V. Results

Our models are able to generate realistic predictions of possible futures for several different tasks, and can use these predictions to make intelligent decisions about how they should move to solve planning problems. See Fig. 1 above for an example: we give the model input images x_0 and x_i, and see realistic results as it peforms three actions: lifting closing the gripper, picking up the green block, and placing it on top of the blue block. In the real data set, this action failed because the robot attempted to place the green block on the red block (next to an obstacle), but here it makes a different choice and succeeds. We visualize the $8 \times 8 \times 8$ hidden layer by averaging cross the channels.

A. Learned Hidden State

First, we explore the meaning of the learned hidden space $\mathcal{H}$. Random samples in the hidden space learned for the block-stacking task (Fig 7, far left) correspond to random parts of objects and robots (Fig. 7, left). To understand what effect $T(h_0, h, a)$ has on $h \in \mathcal{H}$, we randomly sampled a number of hidden states and repeatedly applied $T(h_0, h, a)$ with actions drawn uniformly at random. After 200 steps, we see results similar to those in Fig. 7(right), with objects and the arm positioned randomly in the scene.

B. Model Architecture

We performed a set of experiments to verify our model architecture, particularly in comparing different versions of the transform block T. We compare three different options: the block as shown in Fig. 5, the same block with the skip connection removed, and the same block with the spatial softmax and dense block replaced by a stride 2 and a stride 1 convolution to make a more traditional U-net similar to that used in prior work [10].

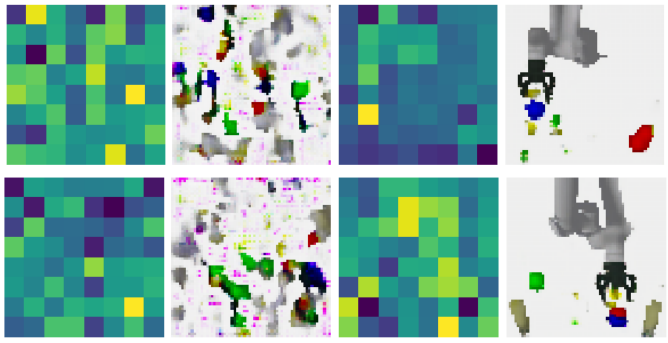

Figure 7: From left to right: (1) randomly sampled hidden state h, (2) $f_{dec}(h)$, (3) h' after 200 $T(h, h, a)$ operations for random a, and (4) $f_{dec}(h')$

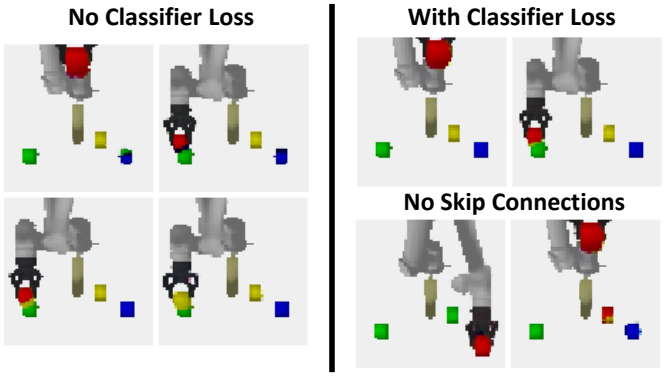

Figure 8: Selected results different possible architectures for the Transform block.

To compare model architectures and training strategies, we propose a simple metric: given a single frame, can we determine which action just occurred? This is computed given the same pretrained discriminator discussed in Sec. III-B. We compare versions of the loss function with and without the classifier loss term, and with this term given one of two possible weights. Both the classifier loss term and the conditional GAN discriminator term were applied to the second of two transforms, to encourage the model to generate predictions that remained consistent over time.

Model	x_1 label	x_1 error	x_2 label	x_2 error
Naive	87.2%	0.0161	74.3%	0.0261
L1	88.1%	0.016	84.5%	0.018
L1+0.01C	87.9%	0.0177	94.3%	0.0214
L1+0.001C	88.2%	0.016	85.4%	0.0184
No Skips	87.5%	0.0224	85.4%	0.0247
cGAN [10]	84.5%	0.0196	77.5%	0.0235

Table I: Comparison of different strategies for learning $T(h_0, h, a)$ as assessed by image prediction error (MAE on pixels) and image confusion on held-out successful examples only.

Model	x_1 label	x_1 error	x_2 label	x_2 error
Naive	89.3%	0.0182	77.3%	0.0276
L1	90.4%	0.0181	86.4%	0.0209
L1+0.01C	90.6%	0.0198	95.6%	0.0239
L1+0.001C	90.9%	0.0181	88.6%	0.0208
No Skips	90.1%	0.0243	89.3%	0.0271
cGAN [10]	86.5%	0.0216	79.9%	0.0260

Table II: Comparison of strategies for learning $T(h_0, h, a)$ as assessed by image prediction error (MAE on pixels) and image confusion on held-out test examples.

Tables I and II show the results of this comparison. There were 37453 example frames from successful examples and 54077 total examples in the data set. In general, the pre-trained encoder-decoder structure allowed us to reproduce high-quality images in all of our tasks. The "Naive" model indicates L1 loss with only one prediction; it performs notably worse than other models due to errors accumulating over subsequent applications of $T(h_0, h, a)$.

The classifier loss term improved the quality of predictions on the second example, and improved crispness of results, at the slight cost of some pixel-wise error on the output images. Here, we see that adding the classifier loss terms (L1+$0.01C$ and L1+$0.001C$) slightly improved recognition performance looking forward in time. This corresponds with increasing image clarity corresponding to the fingers of the robot's gripper in particular. Pose and texture differences largely explain the differences in per-pixel error.

The "No Skips" model was trained the same as the L1+$0.001C$ model, but without the skip connections in Fig. 5. These connections allow us to fill in background detail correctly (see Fig. 8), but were not necessary for the key aspects of any particular action.

The cGAN was able to capture feasible texture, but often missed or made mistakes on spatial structure. It often misplaced blocks, for example, or did not hallucinate them at all after a placement action. This may be because of the noisy data and the large number of failures.

C. Plan Evaluation

Our approach is able to generate feasible action plans in unseen environments and to visualize them; see Fig. 9 for an example. All three traces are generated on the same environment. The first two plans produced by Alg. 1 are recognized as failures, and then the algorithm correctly finds that it can pick up the red block and place it on the blue without any issues. The value function $V(h)$ correctly identified frames as coming from successful or failed trials 83.9% of the time after applying two transforms – good, considering that it is impossible to differentiate between success and failure from many frames. It correctly classified possible next actions 96.0% of the time.

At each node in our tree search, we examine multiple possible futures. This is important both for planning and for usability: it allows our system to justify future results. Fig. 10 shows examples of these predictions in different environments. We see how the system will predict a set of serious failures in the middle row, when attempting to grasp the red or blue blocks, and one possible failure when grasping the yellow block.

We tested our method on 10 new test environments in the stacking task. On each of these environments, we performed a search with 10 samples. Our approach found 8 solutions to planning tasks executing the demonstrated high-level actions, and in 2 tasks it predicted that all of its actions would result in failures, due to proximity to the obstacle. This highlights an advantage of the visual task planning approach: in the event of a failure, the robot provides a clear explanation for why (see the second sample in Fig. 9 for an example).

D. Surgical Image Prediction

We trained our network on 36 examples in the JIGSAWS dataset, leaving out 3 for validation. We were able to generate predictions that clearly showed the location of the arms after the next gesture, as shown in Fig. 2. The learned space $\mathcal{H}$ is very expressive, but loses some fine details such as the thread at times. The result of $f_{dec}(f_{enc}(x))$ still has almost all the same detail as the goal image (right).

Image prediction created recognizable gestures, such as pulling the thread after a suture. While our results are visually impressive, error was higher than in the robotic manipulation task: we saw mean absolute error of 0.039 and 0.062 for generated images x_1 and x_2, respectively. This is likely because the surgical images contain a lot more subtle but functionally irrelevant data that is not fully reconstructed by our transform. It therefore looks "good enough" for human perception, but does not compare as well at a pixel-by-pixel level. In addition, there is high variability on the performance of each action and a relatively small amount of avaiable data.

Again, the cGAN did not have a measurable impact: MAE of 0.039 and 0.067 across the three test examples. As such, the longer training time of the cGAN does not seem justified. In general, it appears to us that conditional GANs are good at modifying texture, but not necessarily at hallucinating completely new image structure.

VI. Conclusions

We described an architecture for visual task planning, which learns an expressive representation that can be used to make meaningful predictions forward in time. This can be used as part of a planning algorithm that explores multiple prospective futures in order to select the best possible sequence of future actions to execute. In the future we will apply our method to real robotic examples and expand experiments on surgical data.

Acknowledgments

This research project was conducted using computational resources at the Maryland Advanced Research Computing Center (MARCC). It was funded by NSF grant no. 1637949. The Titan Xp was donated by the NVIDIA Corporation.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *stat*, 1050:9, 2017.

[2] Michael Beetz, Dominik Jain, L Mosenlechner, Moritz Tenorth, Lars Kunze, Nico Blodow, and Dejan Pangercic. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proceedings of the IEEE*, 100(8):2454–2471, 2012.

[3] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. *arXiv preprint arXiv:1707.09405*, 2017.

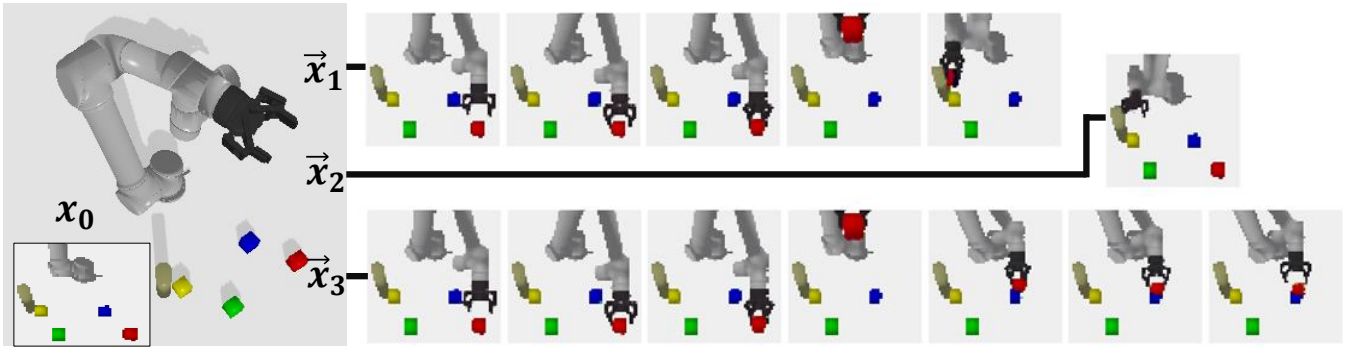

Figure 9: First three results from a call of the planning algorithm on a random environment using Alg. 1. The first two are correctly recognized as failures before the system finds a sequence of actions with high predicted values.

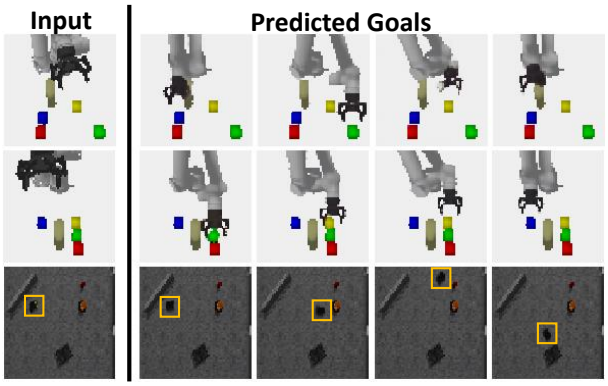

Figure 10: Analyzing parallel possible futures for the stacking and navigation tasks. Top row shows multiple good options for grasping separate objects; the second shows how attempts to grab two objects are clear failures and only one is a clear success.

[4] François Chollet. Keras. https://github.com/fchollet/keras, 2015.

[5] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems*, pages 64–72, 2016.

[6] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *ICRA*, pages 512–519. IEEE, 2016.

[7] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. Deep predictive policy training using reinforcement learning. 2017.

[8] Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David E Smith, et al. PDDL-the planning domain definition language. *http://www.citeulike.org/group/13785/article/4097279*, 1998.

[9] Irina Higgins, Arka Pal, Andrei A Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. DARLA: Improving zero-shot transfer in reinforcement learning. *arXiv preprint arXiv:1707.08475*, 2017.

[10] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016. URL http://arxiv.org/abs/1611.07004.

[11] Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. *arXiv preprint arXiv:1703.06692*, 2017.

[12] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 2015.

[13] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[14] Jue Kun Li, David Hsu, and Wee Sun Lee. Act to see and see to act: POMDP planning for objects search in clutter. In *IROS*, pages 5701–5707. IEEE, 2016.

[15] William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

[16] Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170*, 2017.

[17] Chris Paxton, Vasumathi Raman, Gregory D Hager, and Marin Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. 2017.

[18] Christian Rupprecht, Iro Laina, Robert DiPietro, Maximilian Baust, Federico Tombari, Nassir Navab, and Gregory D Hager. Learning in an uncertain world: Representing ambiguity through multiple hypotheses. *ICCV*, 2017.

[19] Martin E. P. Seligman, Peter Railton, Roy F. Baumeister, and Chandra Sripada. Navigating into the future or driven by the past. *Perspectives on Psychological Science*, 8 (2):119–141, 2013. doi: 10.1177/1745691612474317. URL https://doi.org/10.1177/1745691612474317. PMID: 26172493.

[20] Jaeyong Sung, Seok Hyun Jin, Ian Lenz, and Ashutosh

Saxena. Robobarista: Learning to manipulate novel objects via deep multimodal embedding. *arXiv preprint arXiv:1601.02705*, 2016.

[21] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[22] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, 2015.

[23] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proc. CVPR*, 2017.

[24] S Swaroop Vedula, Anand O Malpani, Lingling Tao, George Chen, Yixin Gao, Piyush Poddar, Narges Ahmidi, Christopher Paxton, Rene Vidal, Sanjeev Khudanpur, et al. Analysis of the structure of surgical activity for a suturing and knot-tying task. *PloS one*, 11(3):e0149174, 2016.

[25] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. In *Advances in Neural Information Processing Systems*, pages 3486–3494, 2016.

[26] Théophane Weber, Sébastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

[27] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *arXiv preprint arXiv:1710.01813*, 2017.