Time Reversal as Self-Supervision

Suraj Nair
Google Brain
nairsuraj@google.com

Mohammad Babaeizadeh
Google Brain
mbz@google.com

Chelsea Finn
Google Brain
chelseaf@google.com

Sergey Levine
Google Brain
slevine@google.com

Vikash Kumar
Google Brain
vikashplus@google.com

***Abstract*—A longstanding challenge in robot learning for manipulation tasks has been the ability to generalize to varying initial conditions, diverse objects, and changing objectives. Learning based approaches have shown promise in producing robust policies, but require heavy supervision to efficiently learn precise control, especially from visual inputs. We propose a novel self-supervision technique that uses time-reversal to learn goals and provide a high level plan to reach them. In particular, we introduce the time-reversal model (TRM), a self-supervised model which explores outward from a set of goal states and learns to predict these trajectories in reverse. This provides a high level plan towards goals, allowing us to learn complex manipulation tasks with no demonstrations or exploration at test time. We test our method on the domain of assembly, specifically the mating of tetris-style block pairs. Using our method operating atop visual model predictive control, we are able to assemble tetris blocks on a physical robot using only uncalibrated RGB camera input, and generalize to unseen block pairs. `sites.google.com/view/time-reversal`**

I. Introduction

Learning general policies for complex manipulation tasks often requires being robust to unseen objects and noisy scenes. While hand-engineered state spaces fail to adapt to these settings, camera images provide a rich and flexible source of sensory information. However, learning from visual inputs present a number of challenges: (1) efficiently exploring the state space, (2) acquiring the suitable visual representation for the task, and (3) learning to execute fine control from dense input. To combat these issues, many methods rely heavily on some form of supervision, either as expert demonstrations, shaped rewards, or privileged state information.

We propose a novel method for gaining self-supervision that operates by exploring outward from a set of goal states, *reversing* the trajectories, and learning a **time-reversal model (TRM)** to predict the reversed trajectories, there by paving a way to return back to the goal state. Specifically, during training we generate data by initializing to some set of goal states, applying random forces to disrupt the scene, and recording the subsequent states. We then train TRM to predict these trajectories in reverse given the current state. At test time (when the goal states are unknown), the trained model can take the current state as input and provide a guiding trajectory of frames leading towards the goal (Fig 1). This guiding trajectory can be used as indirect supervision to generate a low level control policy via any model based or model-free techniques. Incorporation of this indirect supervision allows us to handle complex tasks, as it decouples high level task reasoning from low level control, each of which can be learned separately.

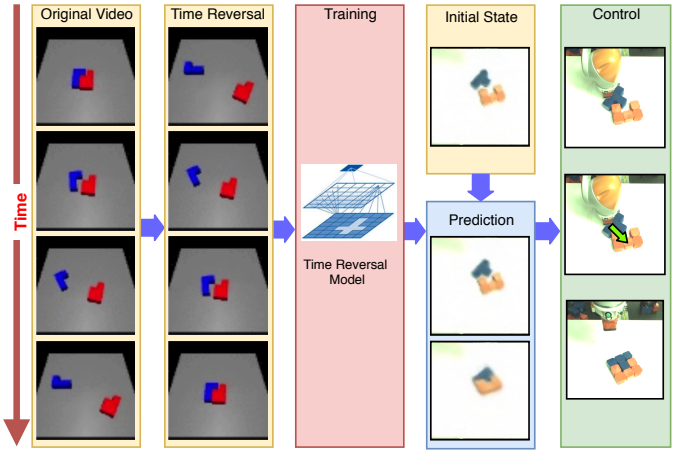

Fig. 1. Our method (1) collects trajectories exploring outward from a set of goal states and reverses them. It then (2) trains a supervised model, the time reversal model (TRM), to predict these reversed trajectories. (3) It uses the trained TRM to predict the trajectory of states leading to the goal state for a new scene. (4) Control is executed to follow the trajectory predicted by TRM.

Most manipulation tasks that one would want to solve require some understanding of objects and how they interact. However understanding object relationships in a task specific context is non-trivial. Consider the seemingly simple task of putting a cap on a pen. Successful task completion is dependent on both concentric alignment of axis and a particular approach direction. Thus we argue that learning the schematic understanding of objects and their relationships for manipulation requires (1) contextual understanding, (2) high level reasoning, and (3) precise control. We manifest these challenges into the problem of tetris-style block mating and attempt to learn the semantic understanding necessary to solve the problem from raw visual inputs.

In particular we tackle the mating of tetris-style blocks sliding along a flat tabletop into a goal configuration (Fig. 2). We train our model using data generated in a MuJoCo simulation [1] with domain randomization [2]. We evaluate our model in both simulation and on a real robot setup with a KUKA IIWA. Experimental results show that TRM can reliably provide supervision towards the goal configuration. In addition, using TRM with a trained visual dynamics model and the cross-entropy method [3] we are able to achieve a 75% success

rate of block pair mating on a physical robot setup using only uncalibrated RGB camera input. The method achieves close performance to far more heavily supervised methods, such as visual model predictive control with ground truth goal and trajectory information. In addition, our method also extends to unseen block pairs with comparable performance, unlike model-free baselines.

Summary of Contributions:

1) Our primary contribution is a novel method for self-supervision that uses time-reversal.
2) We demonstrate that this method generates accurate and valuable guiding trajectories toward the goal, without explicit specification of the goal.
3) We show that our method can be trained in simulation with domain randomization and transferred to the real world.

II. Related Work

Methods for robot control from visual inputs have been demonstrated on problems ranging from driving [4] to soccer [5]. One approach has been visual servoing, which directly performs closed loop control on image features [6]–[8]. While some visual servoing methods work with uncalibrated camera inputs [9], [10], the general hand crafted nature of visual servoing limits the complexity of visual inputs and tasks where it can be applied. Other works have emphasized learning based approaches, in particular the use of deep neural networks to learn visuo-motor control from images [4], [5], [11], [12]. These methods have shown impressive results in the problems of simple manipulation and grasping [13]–[18] by learning task specific policies. While these approaches are generally successful in their task, they have not been demonstrated on more complex tasks, particularly those which require both high-level planning and precise control. These methods have also been shown to work well when trained in simulation with domain randomization and transferred to a physical robot [2], [19]–[21], a strategy which we use in our method.

Model based approaches to robot control have traditionally been most effective in tasks with low-dimensional states, such as helicopter control [22] and robotic cutting [23], however recent methods have found success in learning a dynamics model in image space [24]–[26]. Similarly, Agarwal et al. [27] and Nair et al. [28] have learned inverse dynamics models in image space. These models have been shown to be effective in planning [29], and have even been extended to operate in 3D point cloud space [30], [31]. While these approaches work well on simple tasks, they require additional information during evaluation in the form of either goal images or demonstrations, exactly what our method circumvents.

At the same time, exploration of visual domains remains a significant challenge. A number of recent works have tried to tackle this problem in low dimensional spaces by training goal-conditioned policies, and reformulating seen states as goals as self-supervision, yielding improved sample efficiency [32], [33]. A similar idea has been extended to physical robots and images by Nair and Pong et al [34], who practice reaching imagined goals. This method however still requires goal images at test-time, and tests on a simpler puck-pushing tasks.

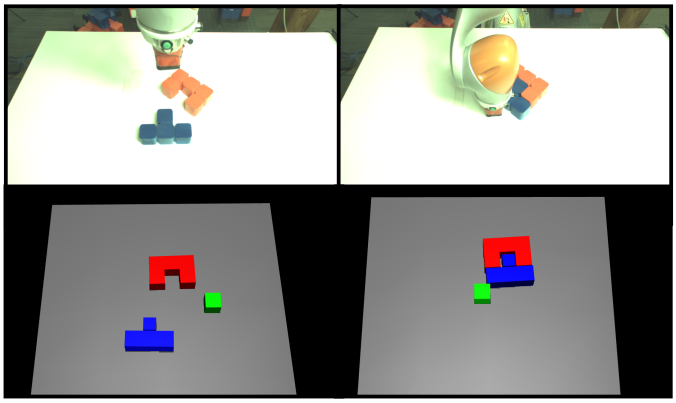

Fig. 2. Environments and Goals: *top row*: the real setup consists of a KuKa IIWA arm mounted over a flat tabletop workspace. *bottom row* the Mujoco simulation environment contains a similar setup of block pairs on a flat table top. The tool in the simulator is a single green block. *left column*: initial states consist of the blocks randomly placed in the scene. *right column*: Goal states consist of any state where the male and female part fit together to complete the 3x3 square. These can occur at any pose - the only criteria to be a goal state is that the parts are properly mated.

Another approach to self-supervised exploration involves resetting to goal states and exploring states around the goal state [35]–[37]. While these methods are most similar to our approach, they do not use the exploration around goal states as supervision or guidance, but rather as additional experience which they add to a policy's replay buffer. As a result these methods still need exploration unlike our method which needs neither goal specification nor exploration at evaluation time. These approaches also have not been shown on physical robots with image input.

III. Preliminaries/Background

We formulate the space of problems in which our method can be applied as finite-horizon, Markov decision processes with sparse rewards. At each timestep t, the agent receives a state $s_t \in S$ and chooses an action $a_t \in A$ based on a stochastic policy $a_t \sim \pi(s_t)$. After taking an action the environment returns the stochastic next state s_{t+1} and reward $r(s_t, a_t) = \mathbb{1}\{s_{t+1} \in S_g\}$ where S_g is a subset of S consisting of all goal states.

Our method is well suited when: **(1)** During a training phase we can reset to some subset of goal states $S'_g \in S_g$, selected at random. **(2)** The Markov chain produced by taking uniformly random actions from any goal state $s_g \in S'_g$ has a non-zero probability of reaching all states $s \in S$. Assumption 1 enables us to reset to goal states during the training phase, but does not provide any general specification of goal states nor any information about how to reach them. Furthermore, assumption 2 simply ensures that all states $s \in S$ can be reached from the goal, a condition satisfied in the vast majority of manipulation problems.

Our formulation does not assume that at evaluation time the objective is to reach a specific state s_g, but rather to reach any

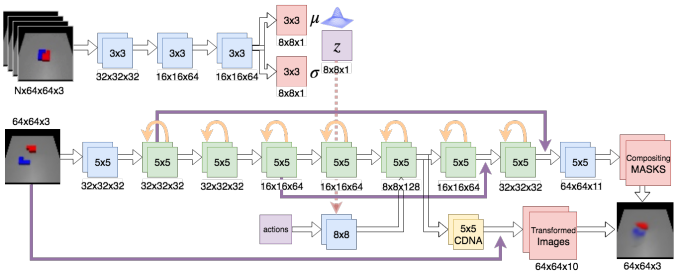

Fig. 3. Architecture of stochastic variational video prediction model (SV2P) [38] used for time reversal and visual dynamics models. The architecture has two main sub-networks, one convolutional network which approximates the distribution of latent values give all the frames, and a recurrent convolutional network which predicts the pixels of the next frame, given the previous frame, sampled latent, action (if available). The code is open sourced in Tensor2Tensor [39] library.

state $s_g \in S_g$, where no specification of the goal is provided. Thus a successful method must be able to (1) reason about what the goal state is given the current state and (2) execute control to reach it.

IV. Method

Formally, our method decomposes the policy $a_t \sim \pi_{\theta,\delta}(s_t)$ into a visual dynamics model $F_\delta(s_t, a_t)$ and time reversal model $TRM_\theta(s_t)$.

The time reversal model, given the current state $s_t \in S$, predicts a sequence of M future states that lead toward a goal state:

$$s^*_{t+1}, s^*_{t+2}, ..., s^*_{t+M} \sim TRM_\theta(s_t)$$

The visual dynamics model simply predicts a sequence of future states from the current state and a sequence of actions:

$$s_{t+1}, s_{t+2}, .., s_{t+N} \sim F_\delta(s_t, a_t, a_{t+1}, ...a_{t+N})$$

At training time both F and TRM are trained in a supervised fashion, and at inference time optimization is used to produce

$$\pi_{\theta,\delta} = CEM(F_\delta, TRM_\theta, s_t)$$

(Fig. 4). This policy $\pi_{\theta,\delta}$ is then what is used to receive states and produce actions at each step of evaluation.

1) Time-Reversal Model Training: TRM is trained to minimize the loss

$$L(TRM_\theta(s_t), [s_{t-1}, s_{t-2}, ..., s_{t-M}])$$

where s_0 is a goal state and $s_0, .., s_T$ represents the sequence of states captured exploring outward from a goal. We train a stochastic variational video prediction (SV2P) [38] model for this task, with the objective of predicting the next $M = 5$ frames given a single input frame (Fig. 3).

2) Visual Dynamics Model Training: The visual dynamics model is another instance of SV2P (with different weights) trained to predict the future sequence of states given the current state and a sequence of actions, minimizing the loss

$$L(F_\delta(s_t, a_t, a_{t+1}, ...a_{t+N}), [s_{t+1}, s_{t+2}, .., s_{t+N}])$$

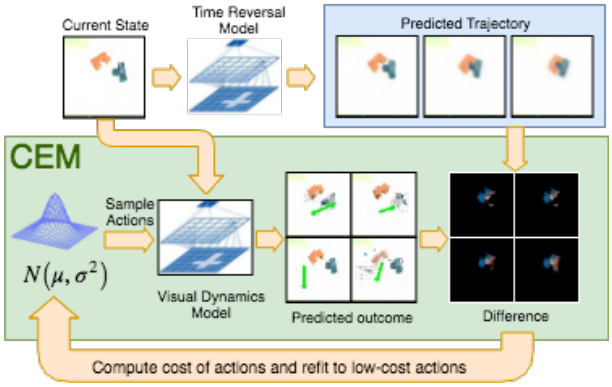

Fig. 4. One step of inference: a new image is provided and is fed through the time reversal model to produce a trajectory toward the goal state. Simultaneously, we use CEM to sample actions and feed them through the visual dynamics model, with the cost of an action defined by its similarity to the TRM trajectory.

A. Inference

At test time, we use TRM's prediction as a guiding trajectory which the visual dynamics model then aims to follow. Specifically, we use the cross entropy method (CEM) to sample actions for which the predicted future images from the visual dynamics model match the predicted trajectory from the time-reversal model. We define one step of the algorithm $CEM(F_\delta, TRM_\theta, s_t)$ in Fig 4.

Specifically we initialize the Gaussian distribution for CEM to $N(0, 0.1)$, sample 200 actions, refit to the 40 actions with lowest cost, and repeat at most 5 iterations before stepping the action with lowest cost in the environment. We use mean squared error between the predicted outcome images and TRM's 5th frame prediction as cost for each action.

V. Experimental Setup

Here we detail our experimental setup, including objects/goals, the observation and action space, as well as our environments and data collection.

A. Objects and Goals

Our problem of block mating consists of a single block pair, decomposed, with each part randomly placed on a flat tabletop. A tool (simulated cube or a robot end-effector) is used to push the objects in the scene. Each block pair consists of two parts which mated perfectly complete a 3x3 square.

The set of goal states S_g consists of all states where the objects are combined to form the 3x3 square. Importantly, the goal configuration is pose invariant. As long as the 3x3 square is completed, the goal configuration is reached regardless of the location on the table or orientation where the completion occurred. Fig. 2 demonstrates examples of initial and goal states.

In our experiments we operate in the simplified setting where one block is initialized randomly in the top half, the other randomly at the bottom half, and to the correct orientation. Once the episode starts the blocks are free to move and rotate.

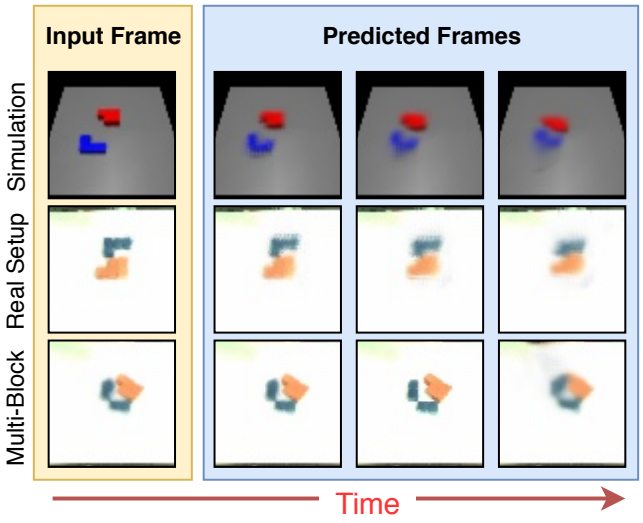

Fig. 5. Time Reversal Model Predictions: Examples of the output of TRM on unseen scenes in both simulation and the real robot setup. TRM takes any state and predicts the sequence of states leading to the goal 3x3 configuration.

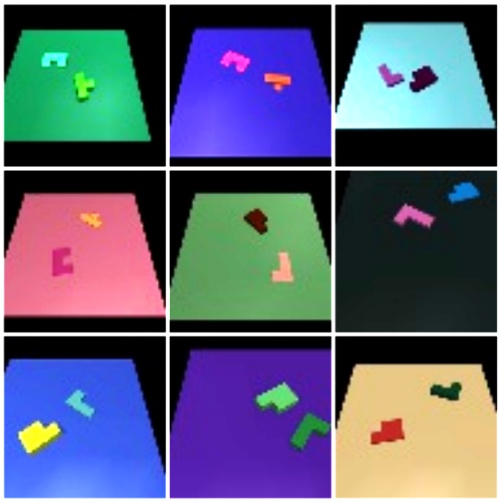

Fig. 6. Domain Randomization: We randomize the color of the blocks, the position of the light, and the position/field of view of the camera to generate a randomized data set. We apply this randomization in the data for both the visual dynamics and time-reversal models, and it allows us to transfer these models a real robot setup without the need for fine-tuning on real data.

B. Observation/Action Space

The observation space consists of 64x64 pixel angled top-down RGB images.

The action space is the high level action space $A \in R^4$ bounded from -0.2 to 0.2, representing the start $< x, y >$ location and an end $< x, y >$ location of a push.

In practice, when an action is called with a start and end point, the robot endeffector (or simulated tool) moves to above the start location, moves down to the table height, pushes linearly towards the end point using Cartesian control with a force threshold, then lifts up and out of the scene, at which point the next state is captured.

C. Environments

We primarily train our models on data generated in a simulation environment, and evaluate our methods on both the simulation environment and a real robot setup (Fig 2)

1) Simulation Environment: The simulation setup is built using the MuJoCo simulation engine [1]. It consists of a flat tabletop, upon which the male and female blocks can slide in the x, y plane or rotate around the z axis. The green cube like tool exists in the scene as well, and executes a push from a goal point to a target point. When it is not pushing, the cube is invisible and out of the scene.

The blocks are composed of between 3 and 5 5cm side length cubes. Each block is free to move from [-17, 17] cm in both the x and y directions.

2) Robot Environment: The robot environment consists of a KUKA IIWA robot operating on a tabletop. The blocks dimensions are the same as in simulation. Again these blocks are free to slide and rotate on the table top. Like the tool in the simulation, the robot end-effector appears to execute a push, lowers itself to the start position, pushes to the goal location, then raises itself out of the scene.

In the robot setup the blocks are free to move [-15, 15] cm in the x direction, and [-14, 16] cm in the y direction, limited due to the physical robot work-space boundaries. Additionally, the blocks in the physical world have less friction than in the simulation, making the physical setup less prone to random, unwanted rotations and forces, hence the task is slightly easier on the physical setup.

D. Data Collection

To train both the visual dynamics model and time reversal model, we primarily collect data in simulation. To transfer to the real world, and to improve overall performance, we apply domain randomization [2]. For the visual dynamics model we also explore collecting data on the real robot.

1) Time-Reversal Model Data Collection: To collect training data for the time reversal model, we collect *disassembly* trajectories. Specifically, in a single trajectory we first initialize to a goal state $s_g \in S_g$ chosen uniformly at random. We then insert random forces to break apart the objects and record the subsequent trajectory of states $s_0(s_g), s_1, ..., s_T$. We then save the trajectory in reverse: $s_T, ..., s_1, s_0(s_g)$

The forces applied to the blocks aim to break apart the blocks, with forces applied outward from the middle of the mated configuration plus some uniformly random noise. In addition uniformly random rotational forces are applied to the blocks, providing more realistic trajectories.

Each trajectory consists of $T = 50$ images sampled at 1 Hz. In total, we collect 20K trajectories. We train SV2P on these sequences which converges with around 100K stochastic gradient decent steps of batch size 16.

2) Visual Dynamics Model Data Collection: To collect training data for the visual dynamics model, we simply execute random actions, and capture the subsequent state. In a single trajectory, we initialize to a state $s_0 \in S$, sample actions

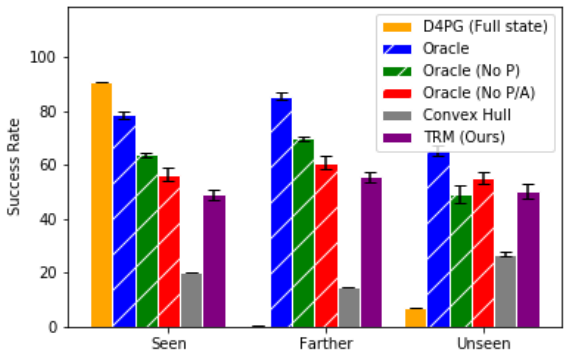

Fig. 7. Supervised Comparison: Results compared to more heavily supervised methods on *seen* blocks, seen blocks initialized *farther* away (at least 30 cm), and on *unseen* blocks. TRM (no supervision), compared to "Oracle (No P/A)" (Groundtruth goal image provided to control policy), "Oracle (No P)" (Closest groundtruth goal image recomputed every step and provided to control policy), "Oracle" (Same as "Oracle (No P)" but including a linearly interpolated path towards the goal image), Distributed Distributional Deep Deterministic Policy Gradient (D4PG) with low level poses as state and sparse reward, and Convex Hull (visual model predictive control with convex hull of the blocks as cost).

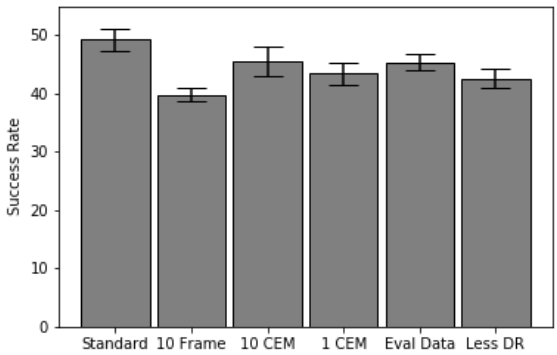

Fig. 8. Ablation Study: We compare the "Standard" algorithm we use to other variants. The "Standard" algorithm uses TRM's 5th frame prediction in computing CEM cost, and does upto 5 steps of CEM, trains on Domain Randomized data, and does not use data from evaluation. We compare this with (1) using 10 frames from TRM instead of 5, (2) doing 10 steps of CEM, (3) doing 1 step of CEM, (4) Finetuning on 1000 trajectories from evaluation, and (5) without camera randomization in the domain randomization process. 500 trials each.

uniformly, and save the transitions (s_t, a_t, s_{t+1}). The actions are limited to 5 cm in size, and are biased towards choosing a goal location within 15 cm of a block 80% of the time to facilitate object interaction.

In total we collect 100,000 trajectories in simulation, where each trajectory consists of a sequence of 10 actions. We train an action conditioned version of SV2P on these sequencues. We explore predicting 2/5/10 frames into the future given the input frame and 2/5/10 actions. We find that predicting 5 frames generally performed the best, and training this takes 300K gradient steps.

3) Visual Dynamics Model Data Collection (Real Robot): Forward model data collection is identical to simulated data collection with the exception that on the real robot there is no bias towards interacting with blocks (since the blocks locations are unknown).

We collected 825 trajectories on the real robot, each consisting of 10 actions.

4) Transfer to the Real World: In our simulated data generation, we explore using domain randomization to enable the learned visual dynamics model and time-reversal model to transfer directly to the physical robot without additional data. In particular, we randomize the color of the blocks and the table and the position of the light by up to 2 meters in any direction. In some cases, we also randomize the position of the camera by up to 10 cm in any direction and randomize the camera's field of view by up to 10 degrees (Fig. 6).

VI. Results

A. Supervised Comparisons

The self-supervised nature of our approach means that most common baselines do not provide a direct comparison. Rather, we assess the performance of our model against methods with substantially more supervision in simulation (Fig 7). Specifically we compare:

Oracle: Replaces the trajectory from TRM with a groundtruth linearly interpolated path from the current image to the nearest goal image, which is recomputed at every step. This represents the maximum level of supervision, and is an upper bound to the performance of our method.

Oracle (No P): Like Oracle, except provides only the goal image itself (recomputed every step), instead of the path of states leading to the goal.

Oracle (No P/A): Like Oracle (No P), except only provides a single goal image for the duration of the episode instead of adaptively updating the goal image.

D4PG: Distributed Distributional Deep Deterministic Policy Gradient algorithm [40], after training for 1 million steps, with ground truth object poses as state and a sparse reward $R(s) = 1$ if the blocks are mated and 0 otherwise.

Convex Hull: Replaces CEM cost with the convex hull of red and blue block. This will give lowest cost when the blocks are mated. It does not require the goal image, but still uses a shaped reward, more supervision than our method needs.

TRM (Ours): Our method using $CEM(F_\delta, TRM_\theta, s_t)$, using the 5th frame prediction from TRM to compute cost and doing max 5 iterations of CEM.

It is important to note that the Oracle comparisons cannot be extended to the real robot, while our method can. We find that despite using substantially less supervision, our method achieves comparable performance to the Oracles. It also outperforms the Convex Hull method, which uses the same visual dynamics model and some supervision in the form of a shaped reward. In addition, we find that while the model-free D4PG comparison is able to learn the standard task reliably, it completely fails when the blocks start farther away (reward is more sparse) and fails to generalize to unseen

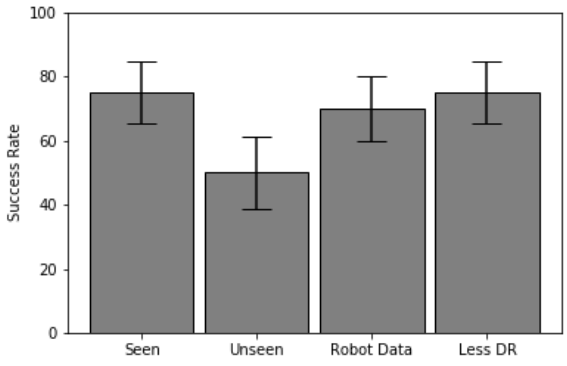

Fig. 9. Robot Results: We test transferring our method to a physical setup with a KuKa IIWA robot. We find that testing on seen blocks, it achieves 75% success. We additionally report the performance on (1) unseen blocks, (2) finetuned on real robot pushing data, and (3) without camera randomization in the domain randomization process. 20 trials each.

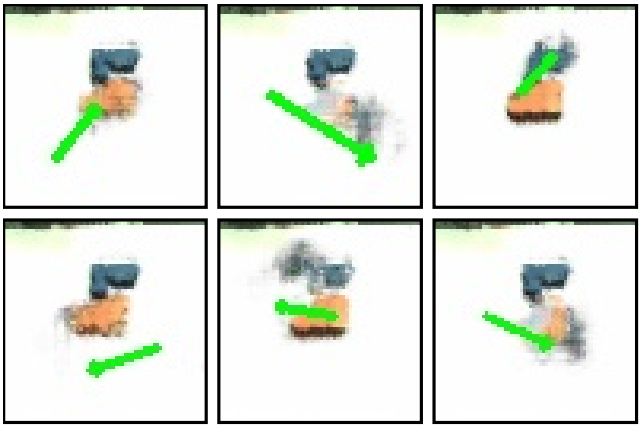

Fig. 10. Visual Dynamics Model Predictions: Examples of the visual dynamics model's predicted next image given different pushes (green arrow).

blocks, despite using a low dimensional state space of object poses (Fig 7).

In addition we explore variations in our method and how they impact overall performance in an ablation study (Fig 8). We observe that performance is improved by taking smaller (5 frame) steps along the TRM trajectory as opposed to larger (10 frame) ones, because actions are less stochastic over small changes. We also see that too many steps of the cross entropy method can lead to worse performance, likely because when the model makes a mistake it collapses into repeating the mistake for the whole episode.

B. Robot Results

We demonstrate that our approach successfully extends to a physical robot setup. In Fig. 9 we report our method's performance on both seen and unseen blocks, as well as with fine-tuning on robot data and a modified version of domain randomization during training that has no camera randomization. Our method successfully mates seen block pairs 75% of the time and unseen block pairs 50% of the time. We also find that fine tuning the visual dynamics model on the 825 robot trajectories and removing camera randomization has no significant impact.

C. Qualitative Results

Here we provide visualizations of the visual dynamics model and time reversal model's performance on unseen data. We see that in unseen settings on both the real robot and in the simulation, the time reversal model predicts frames showing the blocks coming together (Fig 5), even with 3 blocks. We also see that in unseen settings, the visual dynamics model provides frames that accurately represent how the image would change subject to different actions (Fig 10).

VII. Conclusion & Future Work

We have proposed a method which self-supervises task learning through time reversal. By exploring outward from a set of goal states and learning to predict these state trajectories in reverse, our method TRM is able to predict unknown goal states and the trajectory to reach them. This method in conjunction with visual model predictive control is capable of assembling Tetris style blocks with a physical robot and only visual input.

One limitation of our current work is that we only explore the space of problems where the general goal is fixed, and the specific goal state is unknown at test time. In future work we aim to explore using the time reversal model in a larger space of goals where the exact goal specification is provided to the model. Another limitation is that we train our time reversal model in a simulated setup, where we can inject random forces to explore outward from a goal state. Extending this to cases where the agent needs to explore outward is non-trivial, in particular if the agent state is included in the state. We plan to explore this in future work. Lastly, blurriness in the video prediction has made it challenging to extend this work to more complex assembly problems with many objects and complex degrees of freedom. We plan to improve upon the video prediction to produce clearer predictions, enabling more complex tasks.

Acknowledgment

We would like to thank Dumitru Erhan and others from Google Brain Video for valuable discussions. We would also like to thank Satoshi Kataoka, Kurt Konolige, Ken Oslund, Sherry Moore and others from Google Brain who made the experimental setup and infrastructure possible for this research.

REFERENCES

[1] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Oct 2012.

[2] Fereshteh Sadeghi and Sergey Levine. (cad)$ˆ2$rl: Real single-image flight without a single real image. *CoRR*, abs/1611.04201, 2016.

[3] R Y. Rubinstein and D P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. 01 2004.

[4] Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. Morgan-Kaufmann, 1989.

[5] Martin Riedmiller, Thomas Gabel, Roland Hafner, and Sascha Lange. Reinforcement learning for robot soccer. *Autonomous RObots*, 2009.

[6] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation*, 8(3):313–326, June 1992.

[7] K. Mohta, V. Kumar, and K. Daniilidis. Vision-based control of a quadrotor for perching on lines. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3130–3136, May 2014.

[8] W. J. Wilson, C. C. Williams Hulls, and G. S. Bell. Relative end-effector control using cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, 12(5):684–696, Oct 1996.

[9] M. Jagersand, O. Fuentes, and R. Nelson. Experimental evaluation of uncalibrated visual servoing for precision manipulation. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 2874–2880 vol.4, April 1997.

[10] B. H. Yoshimi and P. K. Allen. Active, uncalibrated visual servoing. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 156–161 vol.1, May 1994.

[11] Raia Hadsell, Pierre Sermanet, Jan Ben, Ayse Erkan, Marco Scoffier, Koray Kavukcuoglu, Urs Muller, and Yann LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 26(2):120–144.

[12] S. Lange, M. Riedmiller, and A. Voigtlnder. Autonomous reinforcement learning on raw visual input data in a real world application. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, June 2012.

[13] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015.

[14] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015.

[15] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *CoRR*, abs/1603.02199, 2016.

[16] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. Deep predictive policy training using reinforcement learning. *CoRR*, abs/1703.00727, 2017.

[17] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *CoRR*, abs/1703.09312, 2017.

[18] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arxiv:Preprint*, 2018.

[19] Stephen James, Andrew J. Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *CoRR*, abs/1707.02267, 2017.

[20] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[21] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017.

[22] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems*, pages 1–8, 01 2006.

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

[24] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *CoRR*, abs/1610.00696, 2016.

[25] Alexander Lambert, Amirreza Shaban, Amit Raj, Zhen Liu, and Byron Boots. Deep forward and inverse perceptual models for tracking and prediction. *CoRR*, abs/1710.11311, 2017.

[26] Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *CoRR*, abs/1710.05268, 2017.

[27] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016.

[28] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. *CoRR*, abs/1703.02018, 2017.

[29] Chris Paxton, Yotam Barnoy, Kapil D. Katyal, Raman Arora, and Gregory D. Hager. Visual robot task planning. *CoRR*, abs/1804.00062, 2018.

[30] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. *CoRR*, abs/1606.02378, 2016.

[31] Arunkumar Byravan, Felix Leeb, Franziska Meier, and Dieter Fox. Se3-pose-nets: Structured deep dynamics models for visuomotor planning and control. *CoRR*, abs/1710.00489, 2017.

[32] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017.

[33] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep RL for model-based control. *CoRR*, abs/1802.09081, 2018.

[34] Murtaza Dalal Shikhar Bahl Steven Lin Sergey Levine Ashvin Nair, Vitchyr Pong. Visual reinforcement learning with imagined goals. *arxiv:Preprint*, 2018.

[35] Ashley D. Edwards, Laura Downs, and James C. Davidson. Forward-backward reinforcement learning. *CoRR*, abs/1803.10227, 2018.

[36] Carlos Florensa, David Held, Markus Wulfmeier, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *CoRR*, abs/1707.05300, 2017.

[37] Anirudh Goyal, Philemon Brakel, William Fedus, Timothy P. Lillicrap, Sergey Levine, Hugo Larochelle, and Yoshua Bengio. Recall traces: Backtracking models for efficient reinforcement learning. *CoRR*, abs/1804.00379, 2018.

[38] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. *CoRR*, abs/1710.11252, 2017.

[39] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. Tensor2tensor for neural machine translation. *CoRR*, abs/1803.07416, 2018.

[40] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy P. Lillicrap. Distributed distributional deterministic policy gradients. *CoRR*, abs/1804.08617, 2018.