Object-centric Forward Modeling
for Model Predictive Control

Yufei Ye[1]   **Dhiraj Gandhi**[2]   **Abhinav Gupta**[12]   **Shubham Tulsiani**[2]

[1]Carnegie Mellon University   [2]Facebook AI Research

yufeiy2@cs.cmu.edu   {dhirajgandhi,gabhinav,shubhtuls}@fb.com

https://judyye.github.io/ocmpc/

Abstract: We present an approach to learn an object-centric forward model, and show that this allows us to plan for sequences of actions to achieve distant desired goals. We propose to model a scene as a collection of objects, each with an explicit spatial location and implicit visual feature, and learn to model the effects of actions using random interaction data. Our model allows capturing the robot-object and object-object interactions, and leads to more sample-efficient and accurate predictions. We show that this learned model can be leveraged to search for action sequences that lead to desired goal configurations, and that in conjunction with a learned correction module, this allows for robust closed loop execution. We present experiments both in simulation and the real world, and show that our approach improves over alternate implicit or pixel-space forward models. Please see our project page for result videos.

1 Introduction

What will happen if the robot shown in Fig 1. moves its hand to the right by a few inches? We can all easily infer that this will result in the red block moving right, and possibly hitting the blue one, which would then also move. This ability to perform *forward modeling i.e.* predicting the effect of one's actions is a cornerstone of intelligent behaviour – from realizing that turning the knob opens a door, to understanding that falling off from a height can lead to an injury. Not only does this allow us to judge actions based on their immediate consequences, it also enables us to reason about *sequences of actions* needed to achieve desired goals. As an illustration, let us again consider Fig 1, but now try to imagine how we can get the red block to the right end of the table, but without disturbing the blue object. We know

Figure 1: What happens when the robot arm moves right?

that this can be achieved by first pushing the object up, then towards the right, and then back down. This seemingly simple judgment is actually quite remarkable. In addition to understanding that the naive solution of simply pushing right would not succeed, we also could find, among myriad other possibilities, the sequence of actions that would – by chaining together our understanding of each individual action to understand the effect of the collective. In this work, our goal is to build agents that can exhibit similar capabilities *i.e.* given an image, understand the consequences of their actions, and leverage this ability to achieve distant goals.

This insight of using a 'forward model' to find action sequences that lead to desired outcomes is a classical one in the field of AI, and has been successfully adapted to robotic manipulation tasks [1, 2] for scenarios where the state of the system, *e.g.* shape, pose, mass *etc.* of objects, can be easily represented, and the effect of actions analytically obtained. While this explicit state representation can allow efficient and accurate planning, understanding the state of the system from visual observations or analytically modeling the dynamics is not always possible. Some recent approaches [3, 4] therefore propose to learn a forward model over various alternate representations *e.g.* implicit features, or pixel space. However, we argue that using these implicit or pixel based representations for forward modeling discards the knowledge about the structure of the world, thereby making them less efficient or accurate. When we think of the scene in Fig 1, and the effects of our actions, we naturally think of

the different blocks, and their interactions with each other or the robot. Towards learning forward models that have similar inductive biases, we propose to use a semi-implicit representation – explicitly encoding that a scene comprises of different *entities*, but having an implicit representation to capture the appearance of each entity.

Concretely, we represent each object using its spatial location (in image space) and an implicit feature that is descriptive of its appearance (and can implicitly capture transforms like rotations). We build on this object-centric scene representation and present an approach that learns to model the effect of actions in a scene via predicting the change in representation of the objects present while allowing for interactions between the objects and the robot, as well as among the objects. This object-centric forward model allows us to capture several desirable inductive biases that help in learning more efficient and accurately models – a scene comprises of spatial objects, actions can affect these objects, and the objects can, in turn, affect each other. We show we can leverage our learned model to search for a sequence of actions that would allow us to reach a desired scene configuration from the current input image. However, as the forward model is not perfect, we additionally propose to use a 'refinement' module that can re-estimate the scene configuration in the context of the observed image. This allows us to robustly act in a closed loop manner to achieve desired goal configurations, and we show that our approach improves over previous pixel-space or implicit forward models.

2 Related Work

Learning Video Prediction. While forward modeling aims to predict the future conditioned on an action, a related task in computer vision community is that of simply predicting the future, independent of any action. Several approaches have attempted to predict videos in pixel space [5, 6, 7]. Instead of directly regressing to pixels, alternate flow-based prediction models have also shown promising results [8, 9]. However, these can typically only handle small motions between frames, and need a large number of samples to overcome this inductive bias. Most related to our work is the approach by Ye *et al.* [10], which also pursues prediction in an object-centric space, and in this work we show these can be extended to action-conditioned prediction and planning.

Predictive Forward Models for Acting and Planning. In the robotics community, learned forward models have been used for a plethora of tasks *e.g.* leveraging forward models for exploration [11, 12, 13], or to learn a task policy [14, 15, 16]. More related to ours, some approaches [3, 17] jointly learn a forward and inverse model, where the latter is regularized by the former and can be used to greedily output actions given current observation and a short-term goal. We adopt the philosophy for some recent methods [18, 19] that also tackle longer horizon tasks, by training a forward model and then using a planner to generate actions. However, these methods still face difficulty in handling large change per action or a large number of actions. We overcome these limitations by leveraging object-centric representations for forward modeling and planning.

Structured Models for Physical Interactions. Rather than predicting in implicit or pixel-space representation, a line of work with a motivation similar to ours, models physics by explicitly modeling the state transitions, using known [20] or predicted [21, 22] physical properties. However, for generic manipulation tasks in the real world, the dynamics and physical properties cannot easily be captured analytically. Recent learning-based works [23] overcome this in a data-driven manner, and show impressive results for forward modeling and planning with previously unseen, but isolated objects. Towards handling more generic scenarios, some approaches leverage graph neural networks to reason about the interaction between objects [24, 25, 26, 27], but typically apply their methods to simpler scenarios that do not involve robotic manipulation and where the state can be estimated. Janner *et al.* [28] show that such compositional forward models can be applied for tasks like block stacking, but learn these for predefined high-level action primitives. In contrast, our work targets forward modeling for low-level continuous control, where a long sequence of actions is required to achieve a goal.

3 Approach

Given an image depicting multiple objects on a table, and a goal image indicating the desired configuration, we aim to plan a sequence of pushing actions $a^{1:T}$ such that executing them leads to the goal. To search for an optimal action sequence, a forward model is essential to hallucinate the future configurations that would result from any such sequence. Our insight is that to manipulate
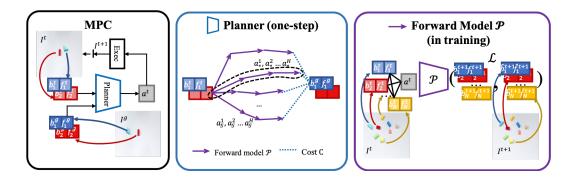
Figure 2: Left: We demonstrate MPC in testing time. Given a goal and initial configuration, the planner takes as input the object-centric representation (section 3.1) and outputs an action to execute. Then, a new observation is obtained to repeat the loop. Middle: Inside the planner, several action sequences are sampled and unrolled by the forward model (section 3.2). The best sequence with respect to the cost is selected, among which only the first action is executed (section 3.3). Right: The forward model $\mathcal{P}$ takes as input a representation of a scene with an action and predicts the next step. It is supervised by the ground-truth representation of the future.

in a complicated environment with multiple objects, object-level abstraction is crucial to make long-horizon predictions and plans. We propose to use an object-centric forward model that can predict the effect of actions via implicitly modeling the interactions among the objects and the robot. While the learned model allows planning using the object-centric representation, our estimate of the objects' locations after a performed action is not perfect and needs to be re-estimated for closed loop execution. We therefore also propose to learn a refinement module to correct our predicted representation using the updated observation.

3.1   Object-Centric Representation.

Given an observation in the form of an image, we aim to predict, in some representation, the effect of actions, and then plan towards desired goals using this learned model. What the form of this representation should be is an open research question, but it should be efficient to learn both prediction and planning with. Our insight is to explicitly leverage the fact that multiple distinct objects are present in typical scenes, and they can naturally be represented as 'where' and 'what' *i.e.* their location and visual description. We operationalize this insight into our representation.

Concretely, given an observed image I^t and (known/predicted) location of N objects $\{b_n^t\}_{n=1}^N$ in the image, we use an object-level representation as $\{x_n^t\}_{n=1}^N$. Each object is represented as $x_n^t \equiv (b_n^t, f_n^t)$, where b_n^t is the observed/predicted location and f_n^t is the implicit visual feature of that object which encodes rotation, color, geometry, etc. The location b_n^t is simply the xy-coordinate in image space. f_n^t is a feature extracted from a fixed sizes window centered on b_n^t, extracted by a neural network with ResNet-18 [29] as backbone.

3.2   Object-Centric Forward Model

Given the current object-centric descriptor $\{x_n^t\}_{n=1}^N = \{(b_n^t, f_n^t)\}_{n=1}^N$ of current time t, and an action about to execute a^{t+1}, the forward model $\mathcal{P}$ predicts the representation $\{x_n^{t+1}\}$ for each object at the next time step $t+1$, i.e. $\{x_n^{t+1}\} \equiv \mathcal{P}(\{x_n^t\}, a^{t+1})$. To predict the effect of a longer sequence of actions $a^{t:t+H}$, we can simply just apply the forward model iteratively H times to hallucinate representation at time $t + H$.

To allow modeling the interaction among robot and objects, the forward model $\mathcal{P}$ is implemented as an instance of Interaction Network[30]. In general, the network takes in a graph (V, E), where each node is associated with a vector. The network learns to output a new representation for each node by iterative message passing and aggregation. The message passing process is inherently analogous to physical interactions. To allow for robot-object and object-object interaction, besides each object represented as a node, the action of the gripper is added as an additional node, with the features being a learned embedding of the action a^{t+1}. In addition to the predictor $\mathcal{P}$, we also train a decoder $\mathcal{D}$ to further regularize features to encode meaningful visual information. Similar to [10], the decoder takes in $\{x_n^t\}$ and decodes to pixels.

3

To train the forward model, we collect training data in the form of triplets (I^t, a^{t+1}, I^{t+1}), where I^t denotes observed images, In addition, we also require location of each object at every time step $\{\hat{b}_n^t\}$ and the correspondence of those objects across time. We argue these annotations (with some possible noise) can be obtained using an off-the-shelf visual detector, as we demonstrate on real robot data in section 4.3. We supervise the model using a combination of two losses – a reconstruction loss and a prediction loss. The reconstruction loss forces the features to encode meaningful visual information (and prevent trivial solutions), and the prediction loss encourages the forward model to predict plausibly both in feature space and in pixel space.

$$L_{recon} = \|\mathcal{D}(\{\hat{x}_n^t\}) - I^t\|_1$$
$$L_{pred} = \|\mathcal{D}(\{x_n^{t+1}\}) - I^{t+1}\|_1 + \|\mathcal{P}(\{\hat{x}_n^t\}, a^{t+1}), \{\hat{x}_n^{t+1}\}\|_2^2$$

where $\hat{x}_n^t$ represents features extracted from I^t at the ground-truth object locations.

3.3 Planning Via Forward Model

Given this learned forward model $\mathcal{P}$, we can leverage it to find action sequences that lead to a desired goal. Specifically, given the goal $\{x_n^g\}$ and current state $\{x_n^0\}$, we generate an action trajectory $a^{1:T}$ such that executing them would lead towards the goal configuration.

We optimize the trajectory by a sample-based optimizer – cross entropy method (CEM) [31]. In CEM, at every iteration, it draws S trajectories of length H from a Gaussian distribution, where H is the planning horizon. The forward model evaluates those sequences by computing the distance of the predicted state $\{x_n^H\}$ to the goal configuration. The best K samples are then selected with which a new Gaussian distribution is refit. The function to evaluate distance of two states / cost of actions is:

$$C = \sum_{n=1}^{N} (\|b_n^H, b_n^g\|_2^2 + \lambda \|f_n^H, f_n^g\|_2^2)$$

After τ iteration of optimization, the trajectory leading to the lowest distance to the goal is returned. Instead of executing the whole sequence of length H, only the first step is actually applied. Then we observe the feedback and re-plan the next action.

In our experiment, $S = 200, K = 10, \lambda = 100, \tau = 3, H = 5$. We sample trajectories in continuous velocity space and upper-bound the magnitude.

3.4 Robust Closed Loop Control via Correction Modelling

We saw that given a current representation $\{x_n^t\}$ and the desired goal configuration $\{x_n^g\}$, we can generate a sequence of action $a^{t+1:t+1+H}$, among which the first action a^{t+1} is then executed at every time step, after which we replan. However, as we do not assume access to ground-truth object locations at intermediate steps, it is not obvious what the new 'current' representation *i.e.* $\{x_n^{t+1}\}$ should be for this re-planning. One option is to simply use the predicted representation $\{x_n^{t+1}\} \equiv \mathcal{P}(\{x_n^t\}, a^{t+1})$, but this leads to an open loop controller where we do not update our estimates based on the new observed image $\hat{I}^{t+1}$ that we observe after our action. As our prediction model is not perfect, such a predicted representation would then quickly drift, making robust execution of long-term plans difficult.

To solve this problem, we propose to additionally learn a correction model $\mathcal{C}$ that can update the predicted location based on the new observation image $\hat{I}^{t+1}$. Denote $I[b]$ as the region cropped on image I specified by the location b. Given the initial crop $\hat{I}^0[\hat{b}_n^0]$ to visually describe the object being tracked, and the predicted location cropped on the new observed image $\hat{I}^{t+1}[b_n^{t+1}]$, it regresses the residual Δb_n^{t+1} to refine, such that $b_n^{t+1} + \Delta b_n^{t+1}$ approximates $\hat{b}_n^{t+1}$ and re-centers the cropped region to objects. We train this model using random jitters around the ground truth boxes on the same training data used to learn the forward model.

4 Experiments

Our goal is to demonstrate that our learned object-centric forward model allows better planning compared to alternatives. To this end, we evaluate our method under both synthetic and real-world settings, and observe qualitative and quantitative improvements over previous approaches.
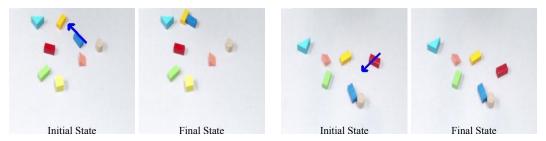
| Initial State | Final State | Initial State | Final State |

Figure 3: Examples of initial state and final state images taken for the push action in real world. The arrow depicts the direction and magnitude of the push action.

4.1 Experimental Setup

Collecting Training Dataset. We work on two pushing datasets, a synthetic environment in MuJoCo [32] and on a real Sawyer robot. To collect training data, multiple objects are scattered on a table. The robot performs random pushes and records the observation before and after. The push action a^t is represented as the starting and ending position of the end effector in world coordinate.

In the synthetic dataset, we generate 10k videos of pushing two randomly composed L-shape objects on the table. Each video is of length 60 (600k pushes in total) and motion between frames is relatively small. To train our prediction model, we extract the ground-truth locations from the MuJoCo state.

To collect real-world data, we generate 5k random pushes (10k images), where the length of each push is relatively large. As a result, in some of these actions, objects can undergo large motion (Figure 3). To obtain the location and correspondence of objects in training set, we manually annotated around 30 images to train a segmentation network [33]. The location is assumed to be the center of the corresponding mask. All of the data collected for the experiment is publicly available at data link.

Evaluation Setup. In both synthetic and the real world, the test set is split into two subsets with one object and two objects, respectively. For quantitative evaluation, we evaluate our model and baselines in simulation, using the distance of objects to the goal position as the metric. In the simulated test set, the distance of initial configuration to the goal is 15 times larger than the length of a single push. The locations are only available to models for the initial and goal configuration, but not at the intermediate steps. In those intermediate steps, only new images are observed and state information is updated and estimated by the models themselves. In the real robot setting, we manually create some interesting cases for qualitative comparisons, such as manipulating novel objects and when the robot has to predict interactions to avoid other objects.

Baselines. We compare our approach of the object-centric forward modelling with the following baselines and their variants:

- Implicit forward model [3] (**Imp-Inv/Imp-Plan**): We follow Agrawal *et al.* [3] and learn a forward model in a feature space where the entire frame is encoded as one implicit feature. Imp-Inv generates actions greedily by the inverse model which takes in current and goal feature; Imp-Plan plans action sequence in the learned representation space.

- Flow-based prediction model [4] (**Flow/Flow-GT**): We follow Ebert *et al.* [4] and learn to predict transformation kernel to reconstruct future frame. In planning, the predicted transformations are applied to designated pixels (location) to estimate their motion. Two flow baselines update the state information by maintaining the probability maps of designated pixels as in previous work. During training, Flow-GT is additionally leverages known object locations during training by supervising the desired transform for the object centre locations.

- **Analytic** baseline: If the exact center of mass is known at each step, a straightforward solution is to greedily push towards goal position. This baseline assumes a naive forward model – the change of location at the next step will be same as the change of gripper position.

4.2 Experiment with Synthetic Environment

We measure the performance across methods by analyzing the average distance of objects from their goal positions. We plot the average distance over time between the current location and the goal in world coordinate in Figure 4. We find the the the 'Imp-Inv' fails to generalize to scenarios when the distance of goal and current observation is much ($\times 15$) farther than that in training set, thereby
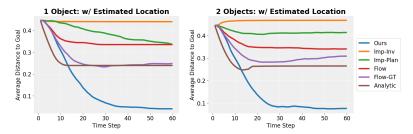
Figure 4: Quantitative results on synthetic dataset: Plot shows the distance between the current configuration and the goal configuration at different time steps with 1 object(left) and 2 objects (right) in the scene.
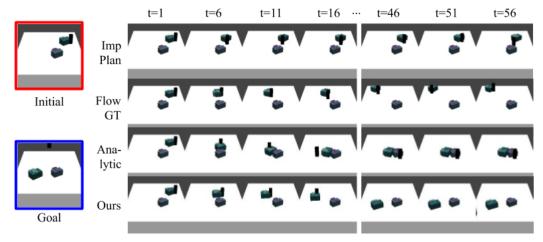

Figure 5: Visualizing an executed action sequence in simulation: Given the initial configuration (in red box) and the goal configuration (in blue box), figure shows the effect of the action predicted by various methods at different time steps. Please refer to the appendix to view all baselines.

showing the importance of planning rather than using a one-step inverse model. The 'Imp-Plan' baseline degenerates significantly for 2 blocks, suggesting a single feature cannot encode the whole scene very well. Flow baseline works much better than Imp-Plan because the motion space is more tractable compared to implicit feature space of frames. Its performance further improves by leveraging location information during training, as seen by the 'Flow-GT' curve. However, using our object-centric model for planning further improves over these baselines as shown in Figure 4.

Figure 5 showcases an interesting example where one block needs to be pushed around the other to reach the goal. In this particular case, learning-based alternatives fail to search a plausible plan. The analytic baseline performs well at the beginning with simple dynamics but loses track of the object when the block collides with the other. In contrast, our approach carries out the correct action sequence and manages to reach the goal, demonstrating that we can reason about interaction among objects. For more qualitative results, please refer to our website.

4.3 Experiments with Real Robot

In the real robot setting, we compare our model with the best performing baseline based on the synthetic results *i.e.* 'Flow-GT'. Figure 6 shows a qualitative result with two blocks. Similar to the example in synthetic data, to push the blue block to the goal position, our model manages to carry out a plan which avoids the red block in between. In contrast, Flow-GT generates relatively random actions, probably because the large motions that can result from a single push are difficult to model. We present additional results in the appendix, and also show that our model can generalize to novel objects by training with simple blocks.

4.4 Ablations

How important is the interaction? We replace the interaction network with a simpler CNN that models independently the effects of action for each object i.e. no interaction. We create a harder dataset in simulation where one block is in the middle of the way for the other block reaching the
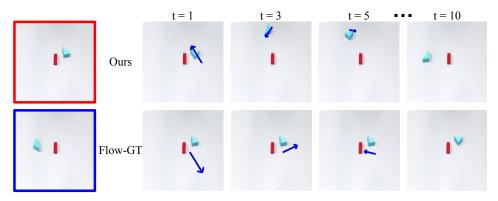
Figure 6: Visualizing an executed action sequence in real world: Given the initial configuration (in red box) and the goal configuration (in blue box), blue arrow shows the sequence of action taken by the robot.
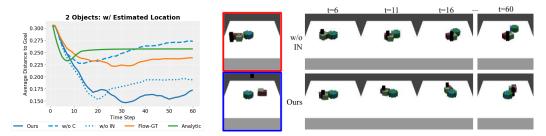

Figure 7: Left: Quantitative result in test subset requiring objects to be pushed around the other. Right: Visualizing an executed action sequence in harder test subset in simulation. The blue block need to be pushed around the other object to achieve the goal.

goal. In this setting, it is more crucial to understand interaction/collision. Figure 7 reports our model, the ablative model (w/o IN) in comparison of two strong baselines. Without IN, the performance is slightly better than our full model at the beginning but degrades more after $T = 30$. This is probably because the model without interaction is more greedy i.e. makes progress initially, but fails to pass around objects. The analytic baseline performs much worse because the simple dynamic cannot estimate the location well since collisions will happen. Figure 7 shows an example of executed actions. Our model can push the object around the other object because it learns a good model of interaction among objects.

Ablating Correction model. We ablate the effect of the correction model using two metrics. First, in Figure 7, we evaluate our model in MPC setting. 'w/o C' estimates location with predicted output without correction and it performs poorly without correction model to close the loop. Secondly, we evaluate it in terms of reducing the prediction error. In Figure 8 (Left), we measure the error between the predicted location and true location, when a 10-step prediction is unrolled with and without the correction module (when using correction module, we use intermediate observation to refine predictions). We see that the prediction error accumulates without any correction.

Lastly, Figure 8 (Right) visualizes some qualitative results. A box around the ground-truth location is plotted as green; the predicted location output by the forward model is plotted as brown; the corrected location is plotted in red. Our model learns to correct the location when the prediction is inaccurate, and retain the predictions when accurate.

Visualizing Planned Action Sequence. We visualize in Figure 9 the $S = 200$ action sequences sampled from the evolving Gaussian distribution across different iterations of the cross-entropy method (CEM) and highlight the $k = 10$ best samples. In the example depicted, we see that after several iterations the model converges to a non-greedy trajectory with the awareness of other objects.

Visualizing Predictions. Figure 10 visualizes the prediction of forward model given the initial configuration x^0, and sequence of one or more actions $a^{1:T}$. In the synthetic data where only small motion happens, both our method and the baseline generate reasonable predictions. However, in the
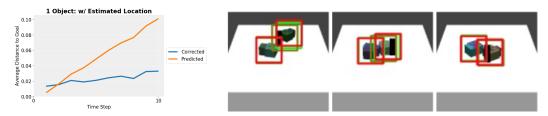
Figure 8: Left: Prediction error with respect to ground truth with or without correction. Right: Visualization of correction model. Ground truth is plotted green; the predicted location from forward model is plotted brown; the corrected one is plotted red.
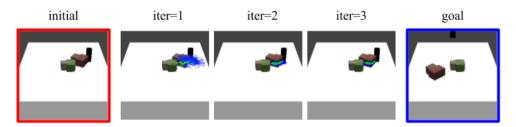

Figure 9: Visualization of sampled trajectories at different iteration of CEM. Trajectories are plotted in blue; elite samples are plotted in green.
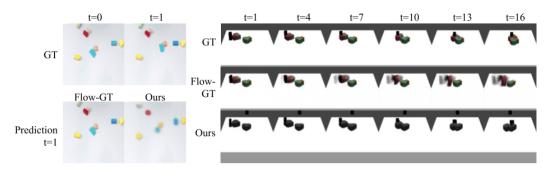

Figure 10: Visualizing prediction of forward model which unrolls the current observation for T steps.

real dataset, the flow baseline cannot learn to predict the flow because the motion is relatively large. In contrast, in the predicted result of our model, when the blue one in the middle is pushed right, the orange one next to it also moves right due to interaction among them.

5   Conclusion

We presented an object-centric forward modeling approach for model predictive control. By leveraging the fact that a scene is comprised of a collection of distinct objects, where each object can be described via its location and visual descriptor, we designed a corresponding forward model that learns to predict in this structured space. We showed that this explicit structured representation better captures the interaction among objects and the robot, and thereby allows better planning in conjunction with an additional correction module. While we could successfully apply our system in both synthetic and real-world settings, we relied on explicit supervision on the object locations during training. It will be an interesting direction to further relax this assumption and let the objects emerge from unsupervised videos. Lastly, while we only modeled the effects of a single class of actions *i.e.* pushing, it would be useful to generalize such prediction to work across diverse actions.

References

[1] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *IJRR*, 1996.

[2] J. Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason. A convex polynomial force-motion model for planar sliding: Identification and application. In *ICRA*, 2016.

[3] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *NeurIPS*, 2016.

[4] F. Ebert, C. Finn, A. X. Lee, and S. Levine. Self-supervised visual planning with temporal skip connections. *CoRL*, 2017.

[5] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *NeurIPS*, 2016.

[6] J. Walker, C. Doersch, A. Gupta, and M. Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *ECCV*, 2016.

[7] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *ICLR*, 2015.

[8] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *NeurIPS*, 2016.

[9] T. Xue, J. Wu, K. Bouman, and B. Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *NeurIPS*, 2016.

[10] Y. Ye, M. Singh, A. Gupta, and S. Tulsiani. Compositional video prediction. In *ICCV*, 2019.

[11] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[12] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *NeurIPS*, 2015.

[13] D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. In *IROS*, 2017.

[14] N. Wahlström, T. B. Schön, and M. P. Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *ICML*, 2015.

[15] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. *ICML*, 2018.

[16] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, 2011.

[17] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *ICRA*, 2017.

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

[19] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *NeurIPS*, 2015.

[20] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik. Learning visual predictive models of physics for playing billiards. *ICLR*, 2015.

[21] J. Wu, J. J. Lim, H. Zhang, J. B. Tenenbaum, and W. T. Freeman. Physics 101: Learning physical object properties from unlabeled videos. In *BMVC*, 2016.

[22] Z. Xu, J. Wu, A. Zeng, J. B. Tenenbaum, and S. Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. In *Robotics: Science and Systems (RSS)*, 2019.

[23] J. Li, W. S. Lee, and D. Hsu. Push-net: Deep planar pushing for objects with unknown physical properties. In *Robotics: Science and Systems (RSS)*, 2018.

[24] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[25] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *NeurIPS*, 2017.

[26] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. *ICML*, 2018.

[27] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *ICLR*, 2016.

[28] M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu. Reasoning about physical interactions with object-oriented prediction and planning. *ICLR*, 2018.

[29] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[30] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *NeurIPS*, 2016.

[31] R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1999.

[32] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

[33] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
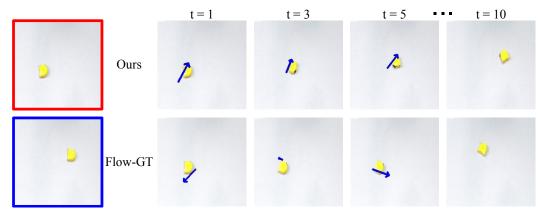
Figure 11: Blue arrow line shows the sequence of action taken by the robot to move objects from start configuration, shown in the red box, to a goal configuration, shown in the blue box.

Appendix A   Real Robot pushing

- Robot Setup:To collect real world data we use Sawyer robot. We place a table in front of it where the objects are placed in order for robot to push it. Kinect V2 camera is rigidly attached overlooking the table for RGB-D perception data. The camera is localized with respect to the robot base via calibration procedure.

- Data Collection Procedure: Given the image I_s of table with object on it, we first perform the background subtraction to get the binary mask corresponding to objects . Using this binary mask, we sample a pixel P_m^C which lies on the object. We treat P_m^C as the mid-point of push. For push start pixel P_s^C, we sample pixel around P_m^C in square such a way that it does not lie on top of the object. The end point of the push P_e^C is calculated based on P_s^C and P_e^C. These pixel location P_s^C, P_e^C in image space are converted to corresponding 3D points P_s^R, P_e^R in robot space using the depth image and camera matrix. Then we use off-the-shelf-planner to move robot gripper finger from $P_s^R \rightarrow P_e^R$. The image I_e is recorded after the arm retracts back. For every push we record the tuple of (I_s, I_e, P_s^R, P_e^R). Figure 3 shows some of the pushing data point collected on real robot. In all we have collected 5K pushing data-points on 8 objects.

- Push novel object: To see how well our method generalizes to novel object, we tested it out for pushing measuring tape. In figure 11 blue arrow shows the push predicted by our method to move it to desired location. Even though our method hasn't seen this object during training of forward model, it is able to push it very close to goal location.

- Flip the object location: To test the effectiveness of our method, we tested it on a bit more challenging scenario. In this case, we have 2 objects on the table. The goal configuration is generated by interchanging the position of objects in start configuration. Figure 12 shows the sequence of action taken by our method to carry out this task.

Appendix B   Baseline Model Details.

- Implicit forward model (**Imp-Inv**) [3]: the model predicts in a implicit feature space where the entire frame is encoded as one implicit feature without further factoring to objects. An inverse model is trained to take in current and goal feature and outputs the action. In testing, the inverse model are applied iteratively to greedily generate action sequence. The inverse model also regularize the forward model to prevent trivial solution.

- Implicit forward model with pixel reconstruction (**Imp-Plan**): The baseline is a variant of Imp-Inv. The action sequences are generated by a planner in the learned feature space. To further regularize the forward model such that it learns a more informative feature space, we train an decoder to reconstruct the frame in pixels. The learned representation of the frame is used by the planner.
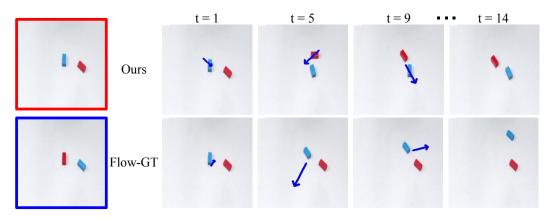
11

Figure 12: Blue arrow line shows the sequence of action taken by the robot to move objects from start configuration, shown in the red box, to a goal configuration, shown in the blue box.
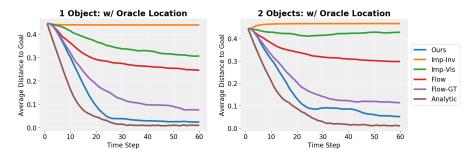

Figure 13: Distance to goal with access to ground truth location at every time step.

- Flow-based prediction model SNA [4] (**Flow**): the model learns to predict transformation kernel to reconstruct future frame. In planning, the predicted transformations are applied to designated pixels (location) to estimate their motion.

- Flow baseline with supervision (**Flow-GT**): The original flow baseline only trains with videos in the unsupervised manner. To leverage the additional information, we provide its variant – besides transforming the pixels, the model also transforms the ground truth location $\{\hat{b}_n^t\}$ to $\{b_n^{t+1}\}$ and minimizes the expected distance of transformed location to the ground truth $\{\hat{b}_n^{t+1}\}$.

- **Analytic** baseline. To leverage the location information, a simple analytic solution is to greedily push in the direction of current goal position to desired position. It assume a simple dynamic – the predicted location at the next step is calculated as the delta position of the gripper.

Appendix C   Plan with Oracle Location

In this part we compare models when we have access to the ground truth location for each new observation at every time step. After every push, the distance between the current location and the goal in world coordinate is plotted in Figure 13. The analytic baseline should converge to zero because the exact center of mass is given by oracle at every time step, hence serves as ceiling performance. The Imp-Inv barely generalizes to scenarios when the distance of goal and current observation is much ($\times 15$) farther than that in training set. Imp-Plan degenerates in 2 blocks settings, suggesting one feature for the whole frame cannot encode complicated scenes very well. Flow works better than Imp-Plan because the motion space is more tractable. Its performance improves in Flow-GT to leverage location information. Our model outperforms other learning-based methods and performs comparably to the ceiling performance (Analytic) without manually specifying pushing toward goal through mass center.
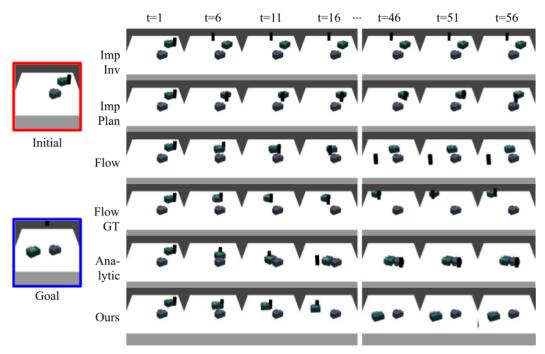
Figure 14: Visualizing an executed action sequence in simulation: Given the initial configuration (in red box) and the goal configuration (in blue box), figure shows the effect of the action predicted by various methods at different time steps. Please refer to the appendix to view all baselines.

Appendix D   Qualitative Results of All Baselines.

In this part we show qualitative results in comparison of all baselines. This supplements Figure 5, which only showcases strong but partial baselines. For more results, please refer to project page.