Science Robotics
AAAS

robotics.sciencemag.org/cgi/content/full/4/26/eaav3123/DC1

Supplementary Materials for

See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion

N. Fazeli*, M. Oller, J. Wu, Z. Wu, J. B. Tenenbaum, A. Rodriguez

*Corresponding author. Email: nfazeli@mit.edu

Published 30 January 2019, *Sci. Robot.* **4**, eaav3123 (2019)
DOI: 10.1126/scirobotics.aav3123

The PDF file includes:

Additional Materials and Methods

Other Supplementary Material for this manuscript includes the following:

(available at robotics.sciencemag.org/cgi/content/full/4/26/eaav3123/DC1)

Movie S1 (.mp4 format). Summary.

Additional Materials and Methods

Reinforcement Learning – Reward and Hyper-parameters

We used the following reward for training:

$$R(o_{1\ldots t_{end}}, a_{1\ldots t_{end}}) = \sum_{i=1}^{t_{end}} (b_1 dx_i - b_2 D_1(q_i, p_i) - b_3 D_2(q_i, p_i)) + b_4 D_3(q_{t_{end}})$$

where $b = (0.1,\ 0.2, 100, 4)$ is a constant vector of gains for the components of the reward function. These values were tuned to the setup through a trial and error procedure to yield converging policies. $b_1 dx_i$ is an accumulating term for positive displacement along the major axis of the block to incentives pushing. In this implementation, we use a linear form for $D_1 = p_i$ over the range of $p \in (0.08, 0.4)$ and zero elsewhere, a fixed value for $D_2 = 100$ for $p > 0.4$, and a fixed value for $D_3 = 1$ if the block is securely grasped by the gripper. The values where found empirically through tuning to yield converging policies. To provide intuition, the 0.4 value of perturbation refers to rotations or displacements exceeding 30 degrees and 20 mm for the entire tower. These large negative costs were necessary to guide the robot away from excessive perturbations due to rewards from positive block motion, else the policy greedily pushes the tower around until collapse, falling short of larger rewards available for a longer game with an intact tower.

The final term denotes a reward over the successful extraction of a block using the gripper. This reward is evaluated per run and summed over the blocks attempted, and each tower in a run is generated at random. A run is ended after a tower topple, if the robot chooses to stop premature to exploring blocks, or all blocks are explored. Empirically we found that policies trained at every 150 samples worked well. Shorter training cycles did not yield convergent policies. Interestingly longer cycles would occasionally yield divergent policies too. This may be attribute to the blame assignment problem.

Bayesian Neural Network Details

The particular numbers where found through a coarse cross-validation across several model structures, including deeper and shallower networks. We found that using tangent hyperbolic nonlinearities in only the first layer; with fully connected remaining layers yielded the best predictive performance. For the inference of weights, we used normal distributions with priors over the means and standard deviations. For the prior over means, we used a zero mean normal distribution with standard deviation of 1, and for the variances we used a half normal distribution with variance of 1. We found empirically that the network performance did not change significantly for variances larger than 1, rather the training time was longer; however, smaller variances would occasionally produce poor predictive models. We hypothesize that this is due to poor mixing at the tails of the narrow normal distributions. To train the models, we used a total

of 200 trajectories (full push sequences) with the Automatic Differentiation Variational Inference implementation of PyMC3.

Model Predictive Controller Details

At each time-step, the controller queries the physics model using samples from the robot action space and computes a cost per action using

$$J_t = \bar{q}_t{}^T Q \bar{q}_t + \bar{a}_t^T R \bar{a}_t + p_t^T T p_t,$$

where $\bar{q}_t$ denotes the current distance of the block to its goal configuration, $Q = diag\{0.3, 0.3, 0.3\}$ denotes the weight matrix penalizing the error in configuration, $\bar{a}_t$ denotes the change in robot action, and $R = diag\{0.1, 0.1\}$ denotes the cost of changing actions. $P = 0.3$ and denotes the cost on perturbing the tower using the proxy value p_t. The first term guides the block to a goal configuration to be grasped. The second term prevents excessive jittering motion of the robot. The last term penalizes the robot for excessive perturbations of the tower. The hyper-parameters were found through extensive empirical analysis. The HMA model has an additional term that penalizes actions that lead to "no move" regimes but does not contain the quadratic penalization term on perturbation.

As a means of controlling the behavior of the robot, we coded a game play module that uses the results of inference and vision system and a finite state-machine to decide on whether to push, extract, place or stop execution. The control loop, together with perception, inference, and planning runs at approximately 0.5 Hz.

Perception – Segmenting Neural Networks

The neural nets were first trained on synthetic renderings of block towers, and then fine-tuned on real block towers with segment annotations. For synthetic data, we rendered 3,000 synthetic images of Jenga tower using Blender. Specifically, each block is a cuboid of size (24.8mm, 75.2mm, 14.3mm). We crawled five images of wood texture from Google Images search and randomly applied them to the blocks. The camera position is a Gaussian distribution in the world coordinate with mean (0.436m, 0.408m, 0.234m) and standard deviation (0.03m, 0.03m, 0.01m). The camera rotation in Euler angles (radian) is also a Gaussian distribution with mean (1.3079, 0.0510, 2.2513) and standard deviation (0.002, 0.002, 0.002). We used a point light source co-located with the camera and the Cycles renderer. For each rendering, its background is randomly chosen from 156 HDR images of outdoor scenes. We emulate the process of a Jenga game by randomly choosing a block from the tower and place it on the top. This process is repeated up to 9 times before we reset the tower. For real data, we collected 150 images of the Jenga tower in different configurations and labeled the mask of each block in each image. We also augmented the data by adding randomly cropped patches of the collected images into the training set; each patch keeps at least 90% of the original image.

Training on synthetic data was conducted on 2 Titan Xp GPUs for 60,000 iterations using stochastic gradient descent (SGD). We used a learning rate of 0.0025, momentum of 0.9, batch size of 2, and weight decay of 1×10^{-4}. We fine-tuned the model on real images for 6,000 iterations using SGD with a learning rate of 2×10^{-5}. All other parameters are the same as those for pre-training.

Perception – Temporal Smoothing

We denote the chamfer distance with (C). In practice, we have $log\ P(m_i|s_i) = 2 \times IoU_s + IoU_b - 5 \times C$. We compute the transition probability as

$$-\ log\ P(s_{i+1}|s_i) = [2 \times (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + \beta(\theta_{i+1} - \theta_i)^2]^{1/2},$$

where $\beta = \max_i \left(1.5, 10 - 0.5 \times \max_i(0, i - 20)\right)$.

Here we represent rotation θ in radian. To speed up inference, we only considered the configurations that are physically plausible, i.e. blocks not penetrating each other. For each time step i, we only consider the 50 candidate states (tuples) that give the highest probability for the observation m_i.