Pose Estimation of the YCB Soup Can in Simulated Scenes as Integrated Discriminative-Generative Inverse Graphics

William Chen (verityw@mit.edu)
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA, 02139, USA

Abstract

For many tasks in modern robotics, manipulation stations must be able to perceive and estimate the poses of objects in an observed scene. While deep, data-driven neural networks have performed well on this front, they largely do not reflect an understanding of causal structures behind an observed scene, as human cognition does. An alternative that addresses this issue is the idea of vision as inverse graphics, wherein a causal generative model of the observed scene is inverted to produce estimates of the scene's underlying parameters. In this project, I explored the use of an inverse graphics system to estimate the pose of the YCB dataset's soup can from images using the PyDrake renderer and physics simulation environment. The renderer was used to generate synthetic, physically-plausible images of the can. Then, a discriminative neural network was used to compute rough object pose estimates. Finally, a generative Monte Carlo Markov chain system stochastically refined these pose estimates in a separate simulated environment using a visual similarity metric. This algorithm makes use of PyDrake's physics engine capabilities to only propose physically-plausible poses for the MCMC process. This system is thus an integration of top-down and bottom-up approaches, where the discriminative subsystem's outputs are improved via a generative subsystem. I finally tested this system on a variety of procedurally-generated scenes of the YCB soup can object in bins, elucidating some of the challenges and advantages of simulation-based inverse graphics systems.

Keywords: Inverse graphics; generative and discriminative models; intuitive physics; pose estimation, robot manipulation; Monte Carlo Markov Chain; simulation engines.

Introduction

Manipulation is an important subfield of robotics. Currently, many applications of robot manipulation – especially in complex tasks involving humans, like cooking or cleaning – require robust localization and perception algorithms, both for the robot to navigate and manipulate the components of their dynamic environment and to understand and model the presence and behavior of human cooperators. To simplify the problem, some have suggested that complicated environments that may have automated components in the future, such as kitchens, will potentially be built in a modular and standardized way, with detailed three-dimensional models to aid robot manipulation algorithms (Fox, 2019).

With the advent of high-fidelity game engines coming from the digital gaming industry, said models have become increasingly useful for tasks like testing manipulation algorithms or easily generating photorealistic labelled training data for neural networks (Guerra, Tal, Murali, Ryou, & Karaman, 2019; Shah, Dey, Lovett, & Kapoor, 2017). The ever-advancing physics simulation abilities of such engines prove useful for many computational cognitive science tasks as well, as they embed intuitive physics into many algorithmic models.

In this project, I will be making use of the Drake physics and graphics simulation engine to explore a simple object pose estimation problem, wherein a soup can model from the Yale-Carnegie Mellon-Berkeley (YCB) object data set is placed in a bin and the system attempts to estimate its pose by just observing an image of the can (Tedrake & the Drake Development Team, 2019; Calli et al., 2015). I will do this by using a discriminative neural network to provide a fast, crude estimate of the can's pose and then use a version of the Metropolis-Hastings algorithm in conjunction with the simulator to refine the initial guess by incrementally altering the estimated pose, rendering the can in the new pose, comparing it to the original image, and then accepting or rejecting the new scene as another pose in the Markov chain with a desirable stationary distribution near the true pose.

Related Works

Generative-Discriminative Integration and Markov Chain Algorithms

(Tu, Chen, Yuille, & Zhu, 2003) formulates the problem of building a "parsing graph" from an image to describe and segment the relations between entities and concepts present in a given scene. To do this, they perform search over the space of possible graphs using a Markov chain algorithm. The authors present a computational framework that integrates generative and discriminative models, wherein the latter computes proposals for the Markov chain and the former defines the target probability distribution for the chain. Compare this to my method described below, wherein the generative subsystem fulfills a similar role of defining the target Markov chain distribution, whereas the discriminative one simply initializes the chain.

Render-and-compare

(Periyasamy, Schwarz, & Behnke, 2019) presents a very similar system to the one described below, likewise being applied to the YCB dataset. The core of the system involves refining initial object pose estimates by rendering the objects, comparing the simulated scene to the observed real one, and making adjustments to the estimates to make the two images more similar. However, two primary differences exist between this system and mine. Firstly, the comparison step is done in a *learned* abstract feature space, rather than with a standard pixel-wise visual similarity metric (described below). Sec-

ondly, their system deploys a differentiable renderer, thereby allowing the iterative changes to the pose estimate to be informed by visual information as well, whereas my system employs a simpler, naive Markov chain method to change and accept adjustments to the estimated object pose. Experimentally, the authors demonstrate that such render-and-compare methods performs well on the YCB Video dataset in a variety of complex scenes, including under different environmental conditions (e.g. lighting) or with object occlusion.

Pose Estimation Networks

Perhaps the best-known YCB pose estimation network is PoseCNN (Xiang, Schmidt, Narayanan, & Fox, 2018), a convolutional network that likewise makes use of intermediary interpretable representations that are fed into subsequent image/object parameter estimates. The network outputs semantic segmentation, bounding boxes, translations, and orientations of YCB objects from RGB camera images. Nevertheless, the network's complexity made it a poor choice for the discriminative subsystem portion of this project, detailed later.

(Wu et al., 2018) presents a simple alternative pose estimation neural network to the ones discussed above. The author presents two networks. The first takes in an RGB image of a scene containing objects of interest and outputs binary masks for said objects, effectively segmenting them from the overall image. Then, these masks are fed into a "pose interpreter network," an extension of the ResNet18 architecture that is trained to compute estimates of the objects' 6-degree of freedom (DoF) poses.

The paper demonstrates that the presented network can generate accurate real-time pose estimates of real-world objects from live video feeds, even when trained with synthetic data. Thus, this network architecture is suitable for the discriminative component of this project, and so I adapted it for use in the pose estimation framework detailed below and trained it on simulated camera data of the YCB soup can.

Approach

Problem Setup

To start, I needed a method of generating a scene of a can in a bin for the following system to estimate the pose of. Using the PyDrake simulation library, I initially created a virtual scene containing an empty bin with a camera/renderer looking down into it. From there, I spawned in an instance of the YCB soup can object at a random position and orientation within the bin. To simplify the problem, the rotation was always a random yaw from the default. As the soup can reference frame's z-axis is perpendicular to its length, this means that the can will always be lying on its side in the bin, rather than upright.

Still, this does not place the can in a physically plausible pose (as the can is spawned slightly above the bottom of the bin, suspended in free space). Thus, I ran the simulation forward by one second, allowing for the can to fall to a physically possible pose. Using the renderer, an RGBA image of the can in the bin is generated and saved. Finally, the ground truth pose of the can is also retrieved using PyDrake commands, and saved as a 7-element list of the form $\hat{x} = [x, y, z, q_w, q_x, q_y, q_z]$, encoding pose information with both a position and quaternion rotation component. The following system will have access to the RGBA image and will try to estimate $\hat{x}$.

It is notationally useful to define $f(x)$ as the mapping from underlying state to RBGA image. This computation is performed when the simulator spawns the can in state x and renders it as an image. The visual pose estimation problem is therefore to find an inverse mapping $f^{-1}(\cdot)$ to go from image to underlying state.

The simulation code is adapted from the lecture notes/homeworks of 6.881 - Robot Manipulation (Tedrake, 2020).

The Discriminative Subsystem

The purpose of the discriminative subsystem is to quickly and roughly compute a pose estimate of the object for the generative subsystem to then refine. I used a simple convolutional neural network for these purposes. Note that such a system does *not* integrate any "physical understanding" in its computations, other than the fact that it is trained on "physically plausible" simulated training data.

Adapted Pose Interpreter Network Architecture I implemented a pose interpreter network heavily based off of the one presented in (Wu et al., 2018). Its overall architecture consists of an initial point-wise convolutional layer followed by a rectified linear unit (ReLU) non-linearity and the ResNet18 (He, Zhang, Ren, & Sun, 2015). This network outputs 1000 classes, which are then connected to a multilayer perceptron, the first layer of which has 256 nodes and is followed by another ReLU non-linearity. Finally, these 256 nodes are connected to two smaller fully-connected layers in parallel. The first has three nodes and estimates the object's Cartesian position. The second has four nodes and estimates the object's orientation as a quaternion. As the latter represents a rotation, it is finally fed through an additional normalization layer that divides by the L_2 length of the 4-dimensional vector containing the outputted 4 elements. In this way, the network effectively estimates the relative magnitudes and signs of the four quaternion elements. See Fig.1 for a full diagram of the network architecture.

Synthetic Data Generation The data generation pipeline is effectively the same as the Problem Setup, described above. The only difference is that the process is repeated and the RGBA images/ground truth soup can poses are saved as labelled training data input/output pairs.

Loss and Training The pose interpreter network described above was implemented in PyTorch (Paszke et al., 2019). Of the three loss functions presented in (Wu et al., 2018), I de-

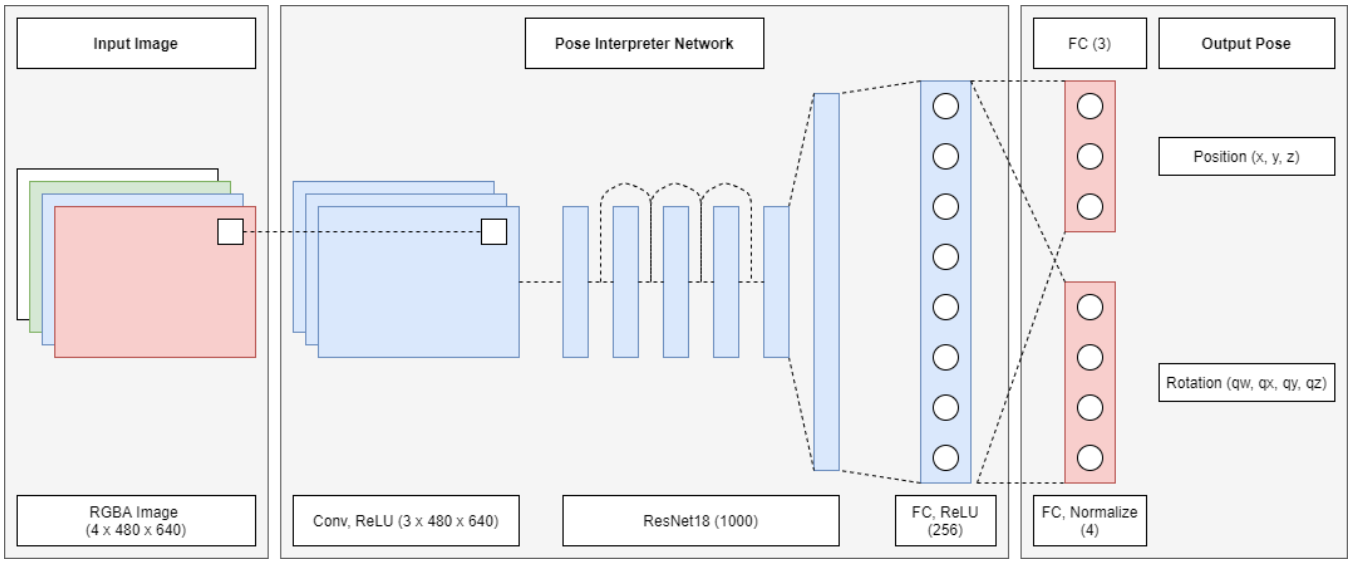

Figure 1: Architecture for the pose interpreter neural network. The first point-wise convolutional layer (with three kernels of size 1×1 and stride 1) serves to lower the number of channels in the 4-channel RGBA image input, as ResNet18 expects a 3-channel image

cided to use the PoseCNN loss, which is of the form:

$$L = |\hat{p} - p| + \alpha(1 - \langle \hat{q}, q \rangle) \tag{1}$$

where p and q are the predicted position and rotation, $\hat{p}$ and $\hat{q}$ are the ground truth position and rotation (the training datum "label"), $|\hat{p} - p|$ is the L_1 distance between predicted and actual positions, $\langle \hat{q}, q \rangle$ is the quaternion inner product of the predicted and actual rotations, and $\alpha > 0$ is some weighing coefficient. The aforementioned inner product can be interpreted as a negative cosine loss between the two quaternions, if they are considered 4-dimensional vectors – since they represent rotations, they are thus of unit length, and so their inner product is the cosine of the hyper-dimensional angle between them.

Note that, when representing rotations as quaternions, q and $-q$ represent the same rotation. Thus, in keeping with (Wu et al., 2018), I decided to always use the quaternion representation with a positive q_w value. In order for the network to learn this, an additional loss term was added to punish negative q_w outputs, resulting in a final loss function of:

$$L = |\hat{p} - p| + \alpha(1 - \langle \hat{q}, q \rangle) + \max(0, -q_w) \tag{2}$$

This concludes the construction of the discriminative subsystem.

The Generative Subsystem

Abstractly, given the image seen by the camera, the generative model should be able to provide and sample from a rough probability distribution over possible soup can poses. This presents an exceedingly computationally intensive problem, wherein the system has to compute a probability density over a large, 6-DoF state space with many non-linear physical constraints. However, the above discriminative approach generates a fast, feed-forward estimate of the soup can's pose from a single RGBA image. This effectively provides a good initial guess for a state near a region of high probability density in the overall distribution, enabling stochastic approximation/sampling methods.

Monte Carlo Markov Chain Methods Monte Carlo Markov Chain (MCMC) methods are often useful for estimating or sampling from probability distributions over intractably large state spaces in a computationally efficient manner. In particular, the Metropolis-Hastings algorithm enables sampling from a target distribution by creating a Markov chain of states whose stationary distribution is the desired one.

Two primary functions are needed for Metropolis-Hastings. The first is a transition function $\pi(x' \mid x)$, which is a conditional probability distribution over x'. The second is a probability ratio function, $\frac{p(x)}{p(x')}$, which gives the *ratio* in probability for two different states in the desired steady state $p(x)$. The algorithm goes as follows. Assume the very first state in the chain is x^0. For the current state x^i, sample a candidate new state x' from the transition distribution:

$$x' \sim \pi(x' \mid x^i) \tag{3}$$

From there, x' is accepted as the next state in the Markov chain with probability given by:

$$\alpha(x', x^i) = \min\left(1, \frac{p(x')}{p(x^i)} \frac{\pi(x^i \mid x')}{\pi(x' \mid x^i)}\right) \tag{4}$$

That is, $x^{i+1} \leftarrow x'$ with probability $\alpha(x', x^i)$, and otherwise, $x^{i+1} \leftarrow x^i$. This process is repeated for many iterations, with

the end result being a Markov chain with the stationary distribution $p(x)$, which can be sampled from by picking spread out states in the chain (after an initial "burn-in" period).

Notably, Metropolis-Hastings does *not* require exact values of $p(x)$, only ratios containing it. This means that it can be run even if only a function that is *proportional* to the true distribution is known; the normalizing partition function, which is often intractable to compute, is therefore *not needed* in such cases. This is critical when using similar algorithms to perform search on the configuration space of the soup can, where the large and complex nature of the set of physically plausible-soup can poses makes computing a closed-form or numerical normalization coefficient impossible. I thus used an algorithm akin to Metropolis-Hastings for pose estimate refinements, after the discriminative network returns an initial rough guess. The system takes in the initial pose estimate and then iteratively transitions to different states in the manner described above, ideally converging to a state close to the observed one.

Physics Engine-based Proposals and Transitions First, a transition function was needed. Since the initial pose estimate from the neural network was usually quite accurate, I decided an appropriate transition would be to either randomly translate or rotate (yaw) the can slightly from the current pose in the Markov chain. These transitions are drawn from zero-centered, low-variance Gaussian distributions, so it is probable that proposed new states are close to the prior one. Note that, since the translations and rotations are drawn from zero-symmetric distributions, the transition probability ratio is always close to 1 (and shall be approximated as such) – e.g. if a transition is proposed moving the can to the right by 1 length unit, then the probability of it being in that new state and moving *left* 1 unit is the same, so the forward and backward transition probabilities are similar ($\pi(a \mid b) \approx \pi(b \mid a)$).

Once a new state was computed, I then used PyDrake to simulate said can pose, in the same way that the data generation system created the network's training data. Critically, when simulating the can in the new state, the simulator would again tick forward 1 second. Retrieving this final pose thus gives the new, proposed state.

Simulating the shifted state achieves two things. First, it ensures the newly proposed state is physically sound, effectively encoding intuitive physics into the inference loop. Second, it enables rendering of the proposed state, which is vital for the next step.

Heuristic-based Similarity Metrics Once the new state has been simulated and rendered, it can finally be accepted or rejected. Ideally, the target stationary distribution of soup can poses that the Markov chain converges to should have a large portion of the probability distribution centered at or very close to the true pose. However, as both subsystems only have an image of the can in the true pose to generate such a distribution, visual similarity thus works as an appropriate heuristic distribution.

Intuitively, the more similar a given state's visual appearance is to the observed image, the more likely it is that said state is close to the underlying actual state, especially in the presence of observational noise. Therefore, the latent probability distribution defined by comparing a given state's image to the observed image of the actual state generally has the desired property where the images that are most visually similar to the observed one also have the can in the most similar pose to the ground truth, therefore making it an appropriate probability distribution to target with MCMC.

Formally, let $S(f_1, f_2)$ be some similarity metric function between two images – the more similar they are, the larger the outputted value. I thus used:

$$\text{Heuristic Probability Ratio} = \frac{S(f(x'), f(\hat{x}))}{S(f(x^i), f(\hat{x}))} \tag{5}$$

in place of $\frac{p(x')}{p(x^i)}$. For a simple similarity metric, I used:

$$S(f_1, f_2) = \exp\{-|f_1 - f_2|_{L_1}\} \tag{6}$$

where $|f_1 - f_2|_{L_1}$ is the mean pixel-wise L_1 distance between the two images.

The Full MCMC Algorithm Starting with the initial pose estimate from the discriminative subsystem x^0, the pose is refined by repeating the following. Given the chain is currently in state x^i, run the simulation forward with the can in said pose and render an image of it $f(x^i)$. Propose a new pose x' and simulate and render the can in said pose, producing image $f(x')$. Calculate the acceptance factor using the similarity metric:

$$\begin{aligned} \alpha(x', x^i) &= \min\left(1, \frac{S(f(x'), f(\hat{x}))}{S(f(x^i), f(\hat{x}))}\right) \\ &= \min\left(1, \frac{\exp\{-|f(x') - f(\hat{x})|_{L_1}\}}{\exp\{-|f(x^i) - f(\hat{x})|_{L_1}\}}\right) \end{aligned} \tag{7}$$

and set $x^{i+1} \leftarrow x'$ with that probability, and otherwise keep it as x^i. After repeating for the desired number of iterations, return whichever pose in the chain resulted in the maximum similarity x^*. This is effectively the MAP estimate of the pose, if one treats the similarity metric as a conditional probability distribution $S(f(x), f(\hat{x})) = p(x \mid f(\hat{x}))$. That is:

$$\text{return } \text{argmax}_{x^* \in \{x^0, x^1, \ldots, x^n\}} S(f(x^*), f(\hat{x})) \tag{8}$$

where $\{x^0, x^1, \ldots, x^n\}$ are the states of the Markov chain. This thus gives a pose that is visually of greater or equal similarity to the initially estimated pose, and thus is ideally closer to the true pose than the one outputted by the discriminative subsystem. This is the final output of the generative subsystem.

Experimental Procedure

To test this pose estimation system, I generated numerous synthetic scenes of the can in different poses, saving both a

render of the scene and the ground truth pose. Then, after passing the render into the discriminative subsystem, I first compared the ground truth pose with the neural network's estimated pose. Subsequently, I inputted that estimated pose as the first state in the generative subsystem's Markov chain in order to refine it. Said refined pose was then also compared to the ground truth, so as to get a numeric metric of refinement. For said comparison metrics, I computed both the L_2 Euclidean distance and the positive difference in yaw.

In terms of system hyperparameters, the discriminative subsystem's neural network was trained for 10 epochs on 4000 synthetic training images and 500 validation images with a batch size of 10 using PyTorch's built-in Adam gradient descent optimizer (Kingma & Ba, 2017). Moreover, for the generative subsystem's MCMC proposal generator, each proposal would be a translation or rotation, with equal probability. For the former, the x- and y-axis displacements from the current state were both drawn i.i.d. from a zero-centered Gaussian with variance .001. For the latter, the change in yaw from the current state was drawn from a zero-centered Gaussian with variance $\frac{\pi}{4}$. The chain itself is 100 states long, with the state with the highest visual similarity metric being returned (as stated prior).

Results

Initially, the generative subsystem did very little to refine the discriminative pose estimate, as the latter was usually very close to the ground truth pose. This meant that all the proposed poses resulted in images that were less similar than the initial guess's rendered image. However, the MCMC algorithm still did improve the pose estimate slightly, as the neural network did not output physically plausible states. To initialize the first state in the Markov chain, the generative subsystem spawns the can in the neural network's outputted pose, runs the simulation forward for 1 second, then saves the new pose of the can in simulation as the first state. Therefore, the can is now lying on the bottom of the bin, meaning it is closer to the true state than just what the neural network outputs. While minor, this does demonstrate how the simulator embeds physics into the pose estimation algorithm, whereas the discriminative subsystem does not.

In a different test, I removed the neural network's orientation estimator, such that the initial discriminative pose estimate *only* outputs an estimated position (the orientation was replaced with a random yaw drawn from a uniform distribution over $[0, 2\pi]$ radians). That way, I could explore if the generative subsystem would be effective at improving the orientation drastically.

As shown in Fig. 2, when the orientation is not estimated by the neural network, the generative subsystem is qualitatively quite effective at refining the neural network's pose estimate. The primary exception to this trend is when the initial pose estimate is very inaccurate, in which case the visual similarity metric is very small. In such situations, even when the Markov chain has new proposed states, said proposals are still generally similar to the first state, meaning the chain does not converge to a distribution centered around the true pose. Essentially, the Markov chain gets stuck at a low-probability plateau in the distribution and the proposal generator does not suggest any sufficiently different states so as to leave that area of the distribution.

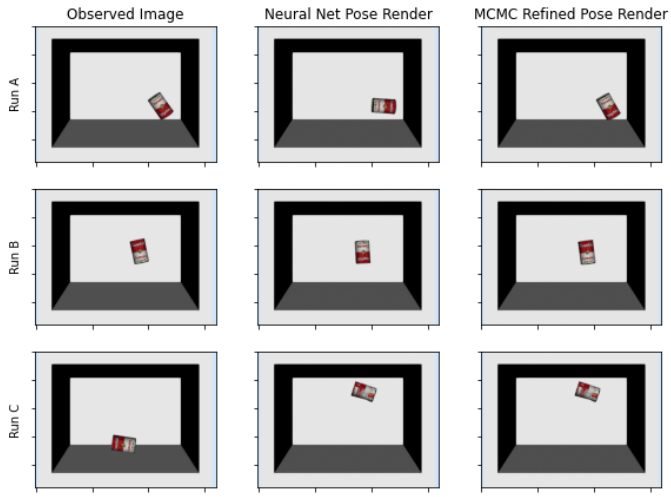

Figure 2: Example renders from three trials. First column is observed image of ground truth render, the second is a render of the can in the neural network's estimated pose (with random yaw), and the third is a render of the can in the refined pose, outputted by the MCMC algorithm. Note the lack of improvement in Run C.

In running 50 trials[1], 8 of them resulted in the initial guess being more than 0.1 units away from the true position, meaning the visual similarity metric was ineffective in the aforementioned way. Of the 42 remaining trials, the average L_2 distance and yaw error between the ground truth pose and the neural network's estimated pose is 0.0055 length units and 1.26 radians respectively. For the MCMC refined pose, those average errors are 0.0079 length units and 0.38 radians respectively.

This shows the effectiveness of the generative subsystem's pose refinements; even when the Markov chain is initialized with a random yaw, it still manages to significantly lower the orientation error. Notably, this is *not* done through a data-driven method – it does not require large quantities of training data to achieve this performance.

Discussion

The above was a very simple toy example to demonstrate concepts in computational cognition and inference, and therefore has many possible avenues for further exploration. A non-comprehensive list of such channels include:

- Adding more/different objects to the bin, thereby also having occlusions, inter-object interactions in simulation, etc.

[1]This was run on Google Colab, so running any more trials would cause the notebook to crash due to using up too much RAM

- Enabling a full 6 degrees of freedom for pose inference, rather than just position and yaw.
- Using other similarity metrics for the generative subsystem's acceptance factor calculation, perhaps including metrics in some learned, high-dimensional feature space.
- Using a neural network as a proposal mechanism for MCMC, rather than the naive method above.
- Using a differentiable renderer to inform the proposal mechanism, effectively formulating the pose estimation problem as a highly-non-linear optimization.

Nevertheless, ultimately, the system succeeded in pose estimation while exploring how renderers/physics engines can be used for inference, how discriminative and generative models can be integrated, and how MCMC methods can be used to both explore and sample from intractably complex probability distributions.

Acknowledgments

While this was not one of the TA-supported projects, I'd like to to thank Joao and Kelsey for their guidance on this project. Likewise, I'd like to thank the staff of 6.881 for helping with troubleshooting the PyDrake simulation setup. Lastly, I'd like to thank my plush wolf, Steven, for providing moral support throughout this project.

References

Calli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., & Dollar, A. M. (2015, Sep). Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, *22*(3), 36–52. Retrieved from http://dx.doi.org/10.1109/MRA.2015.2448951 doi: 10.1109/mra.2015.2448951

Fox, D. (2019, Mar). *Toward robust manipulation in complex scenarios.* Retrieved from https://www.youtube.com/watch?v=Iyje4BSOAxc

Guerra, W., Tal, E., Murali, V., Ryou, G., & Karaman, S. (2019, Nov). Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Retrieved from http://dx.doi.org/10.1109/IROS40897.2019.8968116 doi: 10.1109/iros40897.2019.8968116

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep residual learning for image recognition.*

Kingma, D. P., & Ba, J. (2017). *Adam: A method for stochastic optimization.*

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). *Pytorch: An imperative style, high-performance deep learning library.*

Periyasamy, A. S., Schwarz, M., & Behnke, S. (2019). *Refining 6d object pose predictions using abstract render-and-compare.*

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2017). *Airsim: High-fidelity visual and physical simulation for autonomous vehicles.*

Tedrake, R. (2020). *Robot manipulation: Perception, planning, and control (course notes for mit 6.881).* Retrieved from http://manipulation.csail.mit.edu/

Tedrake, R., & the Drake Development Team. (2019). *Drake: Model-based design and verification for robotics.* Retrieved from https://drake.mit.edu

Tu, Z., Chen, X., Yuille, A., & Zhu, S. (2003, 11). Image parsing: Unifying segmentation, detection, and recognition. In (Vol. 1, p. 18-25 vol.1). doi: 10.1109/ICCV.2003.1238309

Wu, J., Zhou, B., Russell, R., Kee, V., Wagner, S., Hebert, M., … Johnson, D. M. (2018). Real-time object pose estimation with pose interpreter networks. In *Ieee/rsj international conference on intelligent robots and systems (iros).* doi: 10.1109/IROS.2018.8593662

Xiang, Y., Schmidt, T., Narayanan, V., & Fox, D. (2018). *Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes.*