Hardware Acceleration of Robot Scene Perception Algorithms

Invited Paper

Yanqi Liu
Brown University
Dept. of Computer Science
Providence, RI, USA
yanqi_liu@brown.edu

Can Eren Derman
Brown University
School of Engineering
Providence, RI, USA
can_derman@alumni.brown.edu

Giuseppe Calderoni[*]
Politecnico di Torino
Dept. of Electronics and Telecommunication
Turin, Italy

R. Iris Bahar
Brown University
Dept. of Computer Science
School of Engineering
Providence, RI, USA
iris_bahar@brown.edu

ABSTRACT

Hybrid machine learning algorithms that combine deep learning with probabilistic inference techniques provide highly accurate scene perception for robot manipulation. In particular, a 2-stage approach that combines object detection using convolutional neural networks with Monte-Carlo sampling for pose estimation has been shown to perform particularly well under adversarial scenarios. Unfortunately, this accuracy comes at the cost of high computational complexity, which affects runtime, resource utilization, and energy consumption. This paper describes various challenges in developing complexity-aware techniques for robust robot perception and presents a novel hardware accelerator that addresses these challenge. Experimental results show our design is at least 30% faster and consumes 97% less energy compared to an implementation on a high-end GPU. Compared to a low-power GPU implementation, our design is 95% faster while consuming 96% less energy, demonstrating that accurate, energy-efficient scene perception is possible in real time with targeted hardware acceleration.

CCS CONCEPTS

• **Hardware** → **Hardware accelerators**; • **Computing methodologies** → **Rasterization**; • **Computer systems organization** → **Real-time system architecture**.

KEYWORDS

robotics, Monte-Carlo sampling, low-power

[*]G. Calderoni completed this work while a visiting researcher at Brown University

This work is supported by equipment grants from Nvidia and Xilinx Corporations, and a grant through the Brown University Office of Research Development.

ACM Reference Format:
Yanqi Liu, Can Eren Derman, Giuseppe Calderoni, and R. Iris Bahar. 2020. Hardware Acceleration of Robot Scene Perception Algorithms: Invited Paper. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3400302.3415766

1 INTRODUCTION

Scientists are working towards making robots and autonomous systems "taskable"[10] such that they function properly, meet human expectations, and generalize across tasks in various operating environments. The effectiveness of robots to carry out human-requested tasks relies upon their ability to perceive the current state of its world. As robots maneuver around objects and dexterously manipulate them, they must know what objects are in their environment. Robust methods for understanding of objects, as well as their pose and geometry, are essential for robots to reason properly. For standard operating environments, such perceptual understanding remains an open problem that is especially challenging for robustness to complex and dynamic clutter and exacerbated by adversarial obfuscation.

Technological advancements have led to a proliferation of robots using machine learning systems to assist humans in a wide range of tasks. However, we are still far from accurate, reliable, and resource-efficient operations of these systems. Despite the strengths of convolutional neural networks (CNNs) for object recognition, these discriminative techniques have several shortcomings that leave them vulnerable to exploitation from adversaries, such as their need for extremely large training sets, their "black box" decision making, and their inability to recover from incorrect inferences. In addition, the computational, financial, and environmental cost incurred to train these discriminative models can be quite immense, as they often require weeks or even months to adequately train. One recent report estimated that the amount of power required for training and searching a certain neural network architecture used for natural language processing involves the emissions of roughly 626,000 pounds of carbon dioxide [23].

In contrast, generative probabilistic inference techniques are inherently explainable, general, and resilient through the process of generating, evaluating, and maintaining a distribution of many hypotheses representing possible decisions. Unfortunately, this robustness comes at the cost of computational efficiency. Alternatively, our prior work using discriminative-generative approaches (e.g., [12], [2]) offers a promising avenue for robust perception and action. Such methods combine inference by deep learning with sampling and probabilistic inference models, and the ability to represent actual and counterfactual experiments to achieve robust and adaptive understanding. This hybrid approach allows intelligent systems to reason about, interact with, and manipulate objects in complex (and even adversarial) environments.

While our approach does not eliminate the neural network, it does take the burden off the need to massively train a large and complex neural network that may obtain high accuracy, but may still fail under adversarial conditions. Instead, the focus is now on implementing a computationally efficient generative inference stage that can achieve real-time results in an energy efficient manner. In particular, the run time and energy consumption of the generative stage is determined by the range of sampling, the number of iterations, and the computational complexity of the likelihood function. Fortunately, this generative stage turns out to be amenable to various forms of hardware acceleration.

In this paper we offer the GRIP algorithm, introduced in [2] as a discriminative-generative approach for pose estimation, as a robot scene perception approach that can be effectively accelerated in hardware. The generative stage of GRIP takes depth images and performs sample-based generative inference to estimate the pose for each object in a scene. While the GRIP algorithm has been shown to provide high pose estimation accuracy, especially in adversarial environments, it is expensive in terms of runtime, power dissipation, and energy consumption. To address these limitations, in this paper we present a novel, highly parallelized implementation in hardware of the generative stage of the GRIP algorithm. Our approach achieves significant speedup and over 1-order of magnitude energy savings compared to a GPU-based implementation of GRIP. In addition, the algorithm achieves up to 40% improvement in pose estimation accuracy compared to end-to-end neural network approaches, which enables robust performance especially in dark or occluded environments.

The rest of this paper is organized as follows. Section 2 summarizes related work in scene perception, deep learning techniques for object detection and their vulnerabilities, and hardware acceleration techniques for robotics. Section 3 outlines the 2-stage discriminative-generative approach we will be using for robust pose estimation. In Section 4 we describe in detail the novel optimizations we implemented to accelerate the 2^{nd} stage of the algorithm in hardware. Section 5 reports on our experimental results for our FPGA design compared to both a high performance and low-power GPU implementation. Finally, Section 6 contains concluding remarks and suggestions for future work.

2 BACKGROUND

Discriminative-generative algorithms [25], [24], [12] offer a promising avenue for robust perception and action. Such methods combine inference by deep learning with sampling and probabilistic inference models, and the ability to represent actual and counterfactual experiments to achieve robust and adaptive understanding. This hybrid approach allows intelligent systems to reason about, interact with, and manipulate objects in complex (and even adversarial) environments. The value proposition for discriminative-generative inference is to get the best out of existing approaches to computational inference and manipulation while avoiding their shortcomings. Indeed, our prior works using discriminative-generative algorithms have shown promise in terms of improved accuracy and run time [12], [2].

Deep learning methods have been explored in the context of robot manipulation (e.g., [4], [28], [26], [29], [27]). However, there are concerns about the vulnerability of deep networks to adversarial attacks. In the context of robot manipulation, an adversary may not be able to directly alter images observed by a robot, but it can alter the environment from which the robot is capturing its image observations. Similar to adversarial image manipulation, natural clutter could also be slightly and maliciously altered to deceive a CNN used for robot perception. In order to deal with adversarial scenarios, it is advantageous to not rely on adversarial training, which inherently relies on guessing the type of attack beforehand. Rather, we propose a technique that is inherently more robust to unknown attacks during the inference stage since it has some means of recovering from misleading information [12].

Much of the research effort in explainable AI (XAI) has gone into exploring how deep neural networks systems perform when they are operating at their boundary conditions as a means of understanding how they learn (e.g., [16], [22], [30], [1], [13]). For instance, adversarial examples can be used as a tool to answer questions that lead to explanations on how it is making decisions; a user may select an image and then submit it to selected filters to generate foils to see if the classification changes. In this way, the demonstrations of how deep nets can be fooled allow users to understand why it is making certain decisions. Again, while this has led to interesting and useful techniques, it relies on massive training and development of extensive spoofing scenarios to gain an understanding of how the deep neural network is making decisions.

Generative adversarial networks (GANs) [3] have also been proposed to provide additional robustness to attack by trying to deceive the detection network in order to strengthen it. However, the output of such a system still comes from the detection network, which could cause overfitting if the GAN does not sample diversely. In addition, it is still an open problem for how GANs provide interpretability or explainability in results of the detection network. In our approach, it is the generative method that makes the final decision based on input from the discriminative network. This design choice is more amenable to providing greater robustness if we have a suitable model in our likelihood.

Pose estimation is an important step for real-time systems, yet there is little work that considers how it may be accelerated in hardware. Of the proposed approaches found in the literature, they are either not accurate enough for such tasks as robot manipulation (e.g., [19]), provide only partial solutions (e.g.,[17]), or cannot be integrated with a discriminative-generative approach, which is especially useful for reasoning in unstructured environments (e.g., [8]). Particle filtering has been implemented on FPGAs for accelerating object tracking and robot mapping and localization [14], [18], [20], but not for pose estimation.

Recent work proposed a novel FPGA design for 6 degree-of-freedom (6 DoF) object pose estimation based on Monte-Carlo sampling that achieves real time performance with significantly reduced energy consumption [11]. However, this work did not include a critical feature extraction step in the algorithm that takes into account contextual geometric information from 3D point clouds, and therefore cannot obtain high accuracy for certain challenging scenes. This paper extends some of the contributions of [11] by adding a step to exploit point cloud features, streamlining the hardware parallelization, and providing a more comprehensive comparison of performance and energy consumption against prior works.

3 ALGORITHM

Figure 1 shows the two-stage paradigm for the discriminative-generative algorithm proposed in [2]. In the first stage, a CNN takes an RGB image and generates object bounding boxes with confidence scores. In the second stage, the algorithm takes a depth image and the bounding boxes from stage 1 and performs iterated likelihood weighting through generative Monte Carlo sampling to estimate the 6 DoF pose for each object. From this pose estimation, a robot is able to manipulate the object using an appropriate motion planner. Note that the algorithm does not apply any threshold of the confidence scores from the neural network in order to avoid false negative detections generated by the first stage. This is a key takeaway and main advantage of our 2-stage approach; as demonstrated in [12], the 2-stage paradigm increases the possibility of finding the correct pose of the desired object, especially under adversarial conditions such as high occlusion, limited lighting, or altered surfaces that may confuse a CNN. Experimental results reported in [2] showed 25–40% improvement in pose estimation accuracy compared to end-to-end neural network approaches.

While the first stage inference for object detection is quite fast, the second stage is relatively slow and power hungry due to the iterative nature of the sampling process. In particular, the run time and energy consumption of the generative stage is determined by the range of sampling, the number of iterations, and the computational complexity of the likelihood function. Thus, our focus for this paper is on implementing in hardware a computationally efficient generative inference in the second stage that can achieve real-time results in an energy efficient manner. The optimizations we implement to achieve this are described in Section 4; the rest of this section describes the algorithm itself in more detail.

The first stage uses an object detection CNN to generate predictions of object bounding boxes, confidence scores, and class labels. The object confidence score can be represented as the object probability distribution over the observed scene. As the robustness of our approach comes from the generative sampling in the second stage, the network architecture of the CNN itself is not critical. Various network architectures such as VGG [21], ResNet [5], AlexNet [9], and Squeezenet [6] could be used. Our generative sampling algorithm follows the design presented in [24].

Starting the second stage, we use an RGB-D observation that contains an RGB image, Z_r, and a depth image, Z_d. For each object, we define the conditional joint distribution as $P(q, b|o, Z_r, Z_d)$, where q is the 6 DoF pose for the object, o is the class label and b is the bounding box coming from the first stage output. We can formulate the problem as follows:

$$\begin{aligned} &P(q, b|o, Z_r, Z_d) && (1)\\ &= P(q|b, o, Z_r, Z_d)P(b|o, Z_r, Z_d) && (2)\\ &= \underbrace{P(q|b, o, Z_d)}_{\text{pose estimation}}\ \underbrace{P(b|o, Z_r)}_{\text{detection}}. && (3)\end{aligned}$$

We apply iterative likelihood weighting on top of the object detection distribution to perform object pose estimation. Initially, a set of weighted samples $\{q^{(i)}, w^{(i)}, b^{(i)}, z^{(i)}\}_{i=1}^{M}$ are generated, where $q^{(i)}$ represents the 6 DoF pose of the sample object, $w^{(i)}$ the probability, $b^{(i)}$ the bounding box, and $z^{(i)}$ the region of the bounding box. Each sample represents a belief of the object pose over the entire image. A 3D point cloud $r^{(i)}$ is rendered for each sample given the samples' object class o, pose $q^{(i)}$ and corresponding geometric model. For each rendered sample, the weight $w^{(i)}$ is updated to estimate how close the samples' geometry matches to its corresponding observation. The weight of the sample is computed by a likelihood function that takes into account the probability from the CNN output, the raw pixel-wise inlier ratio, and the feature inlier ratio. We define the inlier function using Equation 4:

$$\text{Inlier}(p, p') = \mathbf{I}\left(||p - p'||_2 < \epsilon\right), \tag{4}$$

where $\mathbf{I}$ is the indicator function and p, p' represents a point in an observation point cloud $\mathbf{z}^{(i)}$ and a point in a rendered point cloud $r^{(i)}$. If a rendered point is within a certain distance threshold range ϵ from an observed point, it is defined as an inlier. The ratio of inliers is defined as:

$$I = \frac{1}{|r|} \sum_{a \in z^{(i)}} \text{Inlier}(r^{(i)}(a), z^{(i)}(a)), \tag{5}$$

where a represents the index of a point within observed point cloud $z^{(i)}$ and rendered point cloud $r^{(i)}$ and |r| is the size of the point cloud. We use the same inlier ratio to calculate edge and planar feature inlier ratios.

To better estimate the alignment between sample and observation, we extract geometric features from both the rendered 3D point cloud and the observation point cloud. We use the approach described in [2] where the local surface smoothness can be calculated using Equation 6:

$$c_{(a)} = \frac{|| \sum_{(a') \in \mathrm{N}(a)} \left(\boldsymbol{p}_{(a')} - \boldsymbol{p}_{(a)}\right) ||}{|\mathrm{N}(a)| \cdot ||\boldsymbol{p}_{(a)}||}, \tag{6}$$

where $c_{(a)}$ is calculated by adding all displacement vectors from $\boldsymbol{p}_{(a)}$ to each of its neighbor points $\mathrm{N}(a)$. The value is then normalized by the size of $\mathrm{N}(a)$ and the length of vector $\boldsymbol{p}_{(a)}$.

The weight $w^{(i)}$ of the sample is a linear combination of the confidence score of the bounding box and inlier ratios for raw point cloud I_r, edge features I_e, and planar feature I_p:

$$w^i = \alpha_{box} c + \alpha_r I_r + \alpha_e I_e + \alpha_p I_p, \tag{7}$$

where coefficients α_{box}, α_r, α_e and α_p are empirically determined and add up to 1.

Importance sampling [7] is used to generate a new set of samples based on the current samples' weight distribution. Sample variance

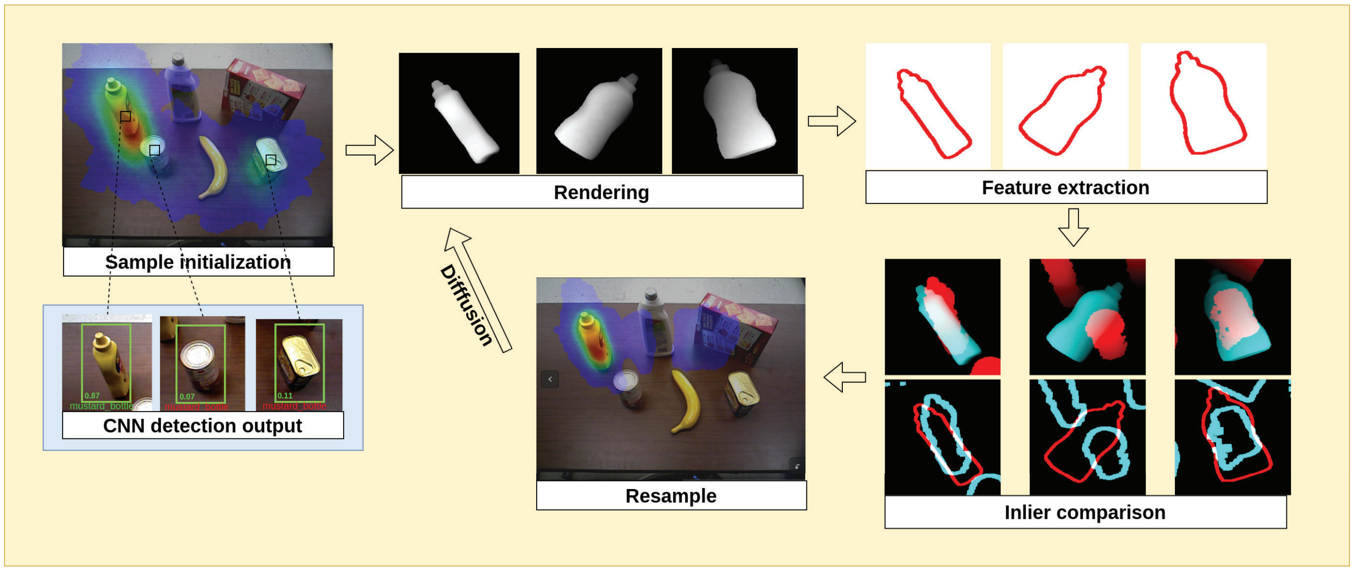


Figure 1: The 2-stage paradigm: The 1st stage uses a CNN for object detection, and the 2nd stage uses the geometric model of the object to perform Monte-Carlo generative sampling.

is increased after sampling by diffusing each new sample pose, $q^{(i)}$, with a Gaussian noise in the space of 6 DoF poses with a small δ:

$$q^{(i)} = (x, y, z, roll, pitch, yaw) + \mathcal{N}(0, \delta). \quad (8)$$

An average sample weight threshold τ is defined to check if the samples have converged. If the weight is above the threshold q^*, corresponding to the highest weight, w^* will be accepted as the sample pose.

The computational bottleneck of the second stage comes from the rendering, point cloud feature extraction, and inlier comparison steps. While each of these steps is amenable to acceleration on a GPU through CUDA programming, together these steps are performed sequentially. In addition, the iterative nature of generative inference and the large amount of samples make the bottleneck more pronounced in its affect on the runtime of the application. To solve the problem, we utilize customized hardware on an FPGA to create a novel implementation that runs feature extraction and inlier calculation simultaneously with rasterization of the object models. Our approach decreases the compute time significantly and introduces a cheap, efficient way of implementing object feature extraction. In addition, the optimized flow drastically reduces the total energy consumption compared to a GPU implementation. The next section describes our novel implementation in more detail.

4 IMPLEMENTATION

Our FPGA implementation aims to speed up the most computationally intensive step in the generative inference algorithm, which is the likelihood evaluation that involves rendering, feature extraction and inlier computation for a large number of samples. Our FPGA implementation consists of 3 main steps:

(1) Rasterize the rendered object using triangles from the object's geometric model
(2) Create a binary bitmap of the edges of the rendered object in parallel with the rasterization process.
(3) Calculate the inlier ratio with rendered depth and extracted edges.

All computation is done in fixed point and we simplify 3D point cloud representation with a 1-D depth representation to further reduce memory and computation resources.

4.1 Rasterization with Feature Extraction

As described in [11], rasterization can be optimized in hardware using multiple customized *raster core* processing units to process samples in parallel. A raster core contains a per-triangle pipeline for rasterization and inlier computation to process all triangles in the object's geometric model. In this work, we extend the raster core unit to include feature extraction of the rendered geometric model. Our new raster core unit is shown in Fig. 2.

For each triangle in the geometric model, we first perform a transformation with the sample's 6 DoF pose. We then perform rasterization for each pixel within the triangle. Given a triangle with three vertices V_0, V_1 and V_2, we first calculate the edge function [15] as given in Equation 9:

$$E_{a,b}(P) = (P.x - V_a.x) \cdot (V_b.y - V_a.y) - (P.y - V_a.y) \cdot (V_b.x - V_a.x), \quad (9)$$

where (a,b) $\in$ [(0,1),(1,2) (2,0)]. To determine if a pixel P is within a triangle, we evaluate E as follows:

- if $E_{0,1} > 0$ and $E_{1,2} > 0$ and $E_{2,0} > 0$, then P is in the triangle
- if $E_{0,1} = 0$ or $E_{1,2} = 0$ or $E_{2,0} = 0$, then P is on a triangle edge
- else, P is outside the triangle

If a pixel is within the triangle, we rasterize it. If a pixel is on the edge of the triangle, we mark it as a temporary edge on the edge map, as shown on the left image of Figure 3(a). The general idea is to combine the rasterized triangles with adjacent edges to create

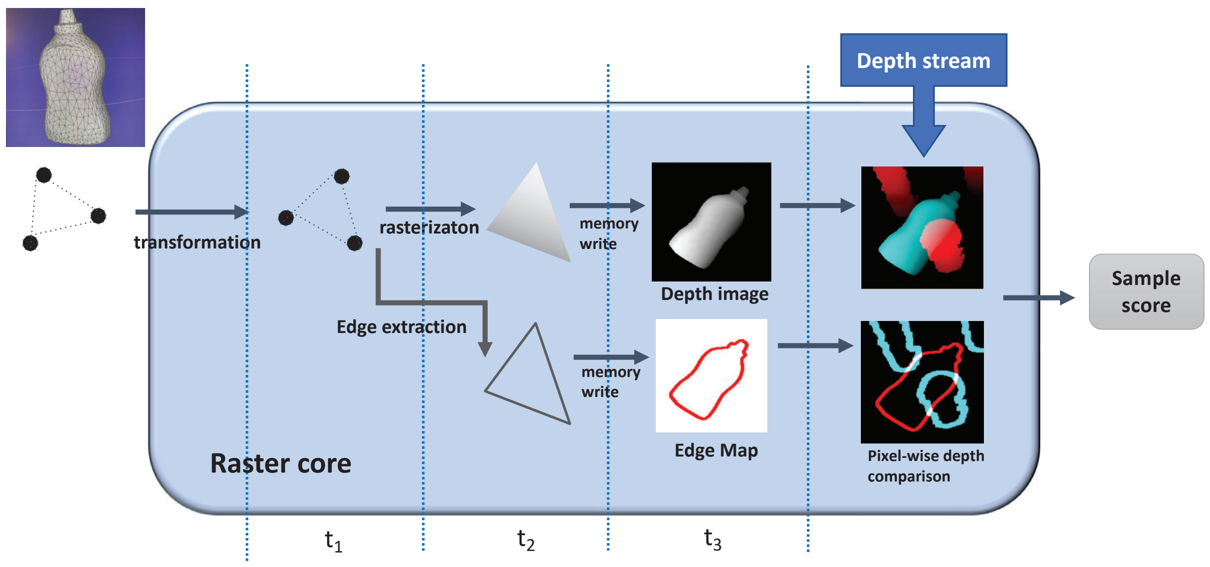
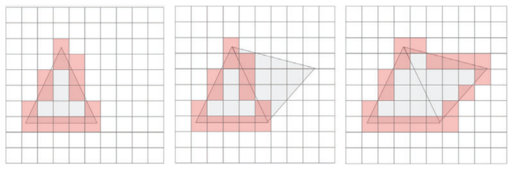


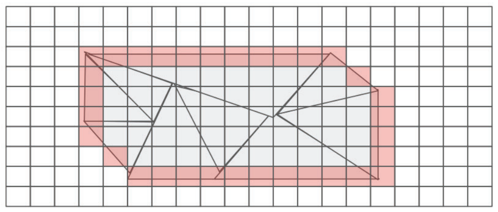
Figure 2: The raster core design. Triangle raterization is in parallel with edge feature extraction. The raster core implements a pipeline for transformation, feature extraction and edge map construction. Multiple raster core units can run in parallel to process different samples.

a new polygon until all triangles within the geometric model are rasterized. Then, the edges of the polygon are used to represent the contour edges of the object. This idea is visualized in the series of images in Figure 3(a) and (b). Examples of rasterization and edge extraction for various objects are shown in Figure 4. Each sample is rasterized within a 200 × 200 frame. Different from [11], we don't keep the partial observation depth in the raster core memory. Instead, we save the rasterized depth image along with a 1-bit 200 × 200 edge map indicating the edge pixels in the rasterized image. After rendering is complete, we stream the observation depth from the depth distributor as described in [11] and compute the inlier ratio for raw rendering depth and rendering edge features.

The original feature extraction step described in [2] calculates the local surface smoothness using Eqn. 6. The method requires memory accesses to neighboring pixels for every pixel and thus generates repetitive memory reads. Furthermore, feature extraction is computed in series with the rendering and inlier steps. Our novel feature extraction implementation in hardware exploits the existing logic used for the rasterization process such that edge information can be obtained in parallel with rasterization with minimal additional computation. In addition, our feature extraction uses Boolean logic operators exclusively, with no algebraic computation.



(a) Filtering coinciding edges among rasterized triangles



(b) End result contour edge map of a box laying on its side overlaid with rasterized triangles

Figure 3: Illustration of the edge detection process.The red box represents the edge pixel

4.2 Sorting, Sampling, and Diffusing

Importance sampling described in Section 3 is used to generate a new set of samples based on the current samples' weight distribution. We follow the implementation described in [11] to reduce memory accesses by using a separate memory to store a set of threshold values to avoid the need to sample all weights explicitly. In this way, the array storing the cumulative density function of sample weights is first searched using coarse threshold steps until the desired range is identified, and then one-by-one within the threshold region until the targeted sample weight is found. Sampling is made more efficient by sorting the samples according to their weights so only the samples with the larger weights will be marked for resampling. For the diffusion step, we use the ping-pong

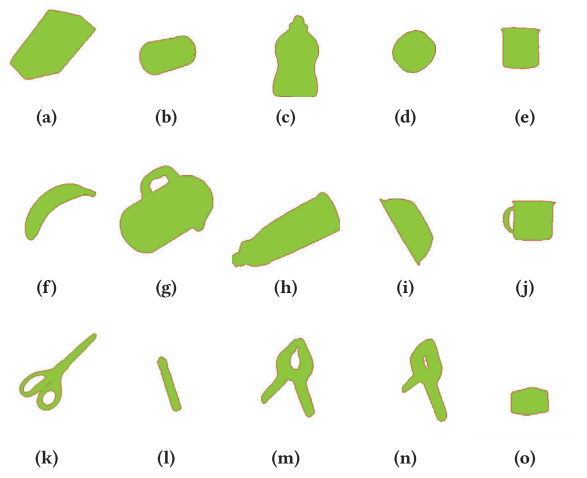


Figure 4: Examples of the drawn edge maps with red and object maps with green of each of the 15 object classes: (a) cracker box, (b) tomato soup can, (c) mustard bottle, (d) tuna fish can, (e) potted meat can, (f) banana, (g) pitcher base, (h) bleach cleaner, (i) bowl, (j) mug, (k) scissors, (l) large marker, (m) large clamp, (n) extra large clamp, (o) foam brick. The green area is the rasterized pixels and the red line is the edge feature pixels

buffer as described in [11], reading the new sample indices generated from the resampler from one memory buffer, adding Gaussian nose to the 6 DoF pose, and writing this new sample to a second memory buffer.

With the resampling and diffusion steps efficiently implemented on the FPGA and a new raster core unit design, we have a full generative sampling flow on the FPGA that is able to perform accurate pose estimation in adversarial environments and achieve fast runtime and efficient power/energy consumption. In the next section we report on our experimental results.

5 EXPERIMENTS

We implemented our Monte-Carlo generative sampling algorithm on a Xilinx Virtex UltraScale ZCU102 board using Vivado HLS high-level synthesis tools. Working within the memory and compute resource constraints of the FPGA, we were able to create a 10-raster core design to process 620 samples clocked at 200MHz. For evaluation, we compare the performance of our FPGA design against designs using two GPU-CPU platforms: 1) an Nvidia Titan Xp (1.4GHz) and Intel Xeon E5 (3.0GHz) platform, and 2) an Nvidia Jetson TX2 (854MHz) with a quad-core ARM A57 (1.2GHz) platform. Below we describe the reference GPU design in more detail and report results in terms of accuracy, run time, resource usage, and energy consumption.

5.1 GPU reference design

Following a similar approach outlined in [2], we implemented a GPU-CPU hybrid design that runs sample initialization, resampling and diffusion on the CPU since these are sequential operations that map more naturally to a general purpose processor. Sample rendering, feature extraction, and inlier computation are accelerated through the GPU. We use OpenGL for rendering of the samples' geometric model and for programming the CUDA cores to perform the inlier computation. We create a kernel to process every pixel distance comparison concurrently. Given the high memory access bandwidth of the GPU, we keep one copy of observation depth in GPU main memory for each sample to access. The GPU transfers the samples' inlier score to the CPU through the PCIe connection where it continues with the resampling and diffusion steps. The data transfer between GPU and CPU is expensive and therefore, we also created a full-GPU implementation to eliminate the transfer time. Specifically, we moved the resampling and diffusion steps to be computed on the GPU and program CUDA kernels to process resampling and diffusion in parallel for each sample.

Table 1: Runtime comparison of complete flows

Implementation	potted meat_can	banana	mustard bottle	cracker box	tomato soup_can
Titan$_{CPU-GPU}$	124 ms	105 ms	144 ms	188 ms	114 ms
Titan$_{all-GPU}$	95ms	87ms	120ms	150ms	90ms
Jetson$_{CPU-GPU}$	1775ms	1539ms	1978ms	2830ms	1734ms
Jetson$_{all-GPU}$	1315ms	1240ms	1478ms	2302ms	1253ms
FPGA (ours)	**72 ms**	**66 ms**	**72 ms**	**100 ms**	**72 ms**

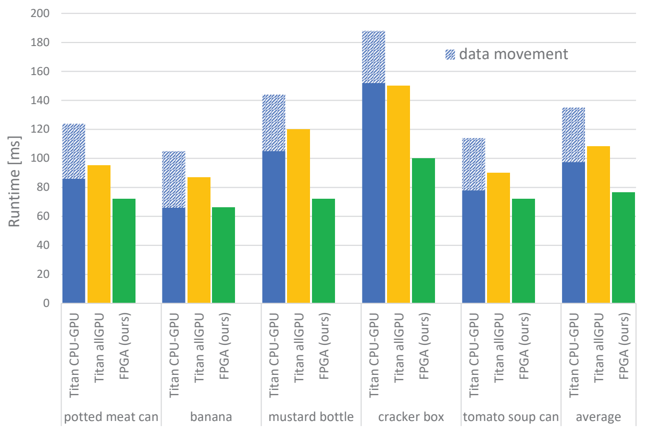
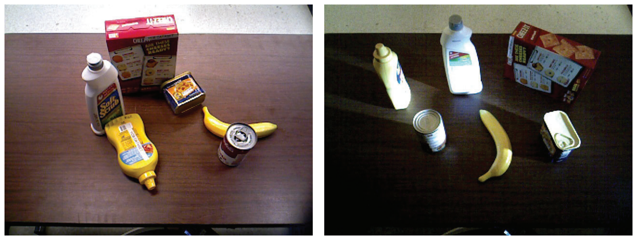


Figure 5: Aveage per-iteration runtime for each object class.

5.2 Speed

The runtime comparisons of our FPGA implementation against CPU-GPU and full-GPU implementations on the two different GPU platforms are shown in Table. 1. Compared to the CPU-GPU and full-GPU implementations running on the Titan Xp, our FPGA implementation achieved on average a 1.76X and 1.42X speedup respectively. Running GPU-CPU and full-GPU implementations on a Jetson board, on average we achieved a 26X and 20X speed respectively. It should be noted that the total runtime for CPU-GPU implementation includes time spent transferring data to/from the CPU and GPU. As illustrated in Figure 5, on average, 33% of CPU-GPU runtime was spent on data transfers on the GPU (as highlighted by the hashed blue bars). The full-GPU implementation eliminates the data transfer inefficiency between CPU and GPU, but still falls short of the runtime from our FPGA implementation, despite the fact that the CPU and GPUs have faster clock rates and larger memory and resource capacities. Objectively, the main benefits of using an FPGA is the configuration of all the resources in close proximity on the same fabric and pipelining data processing across the various steps of the algorithm. This results in less time spent for data transfers, less need to store intermediate results between steps, and more opportunity for parallel execution. Indeed, in our experiments we found that our FPGA implementation achieves significant runtime advantages over the GPU implmentation.

5.3 Accuracy



(a) Occluded Scene (b) Dark Scene

Figure 6: Examples of Occluded and Dark Scenes

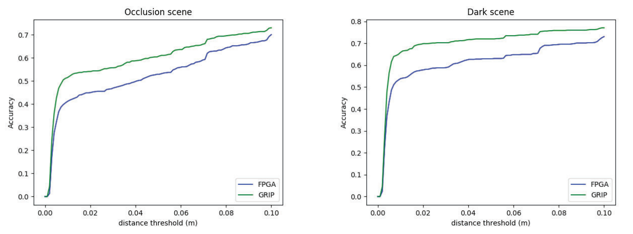
To evaluate the algorithm accuracy, our benchmarking dataset contains 18 RGB-D images taken from a Kinect camera with objects from the YCB dataset [29]. Each scene contains 5–7 different objects that are either placed in close proximity in order to cause object occlusions or taken with limited lighting. Examples for occluded and dark scenes are shown on Figure 6. The pose accuracy for the inferred objects are calculated by average distance difference between predicted pose and ground truth pose. We use ADD and ADD-S as defined in [29] for non-symmetric and symmetric objects respectively.

Figure 7 compares the average accuracy distance-threshold of the five objects for (a) occluded, and (b) dark scenes for our FPGA implementation, and the GRIP implementation of [2]. For each plot, the vertical axes represents the percentage accuracy, while the horizontal axes reflect the average distance threshold between the predicted pose to the actual pose. The area under the accuracy-threshold metric is used for determining the success of a system.

We can see from the figures that the proposed approach achieves approximately the same performance for the dark and occluded settings. Although the accuracy of GRIP is slightly better for smaller distance thresholds, we see that the average accuracy reaches the same level in precision to that of GRIP for higher distance-threshold values. Note that the randomness of the iterated likelihood evaluation results in each flow outputs slightly different results for the same objects in the same image during different trials. Thus, this also leads to a slight difference in accuracy between the two approaches.

5.4 Resource Usage

As mentioned in Section 2, the work of [11] proposed an FPGA design for pose estimation based on Monte-Carlo sampling, but left out the feature extraction step. In Table 2 we compare the single raster core resource usage of our proposed approach to the one implemented in [11], which we refer to as MC_{FPGA}. The additional BRAMs in our design are used for the 1-bit edge map and extra DSPs are added for memory addressing for edge map construction. More FFs and LUTs are needed for additional data streams and edge map logic operations. The lack of a feature extraction step leads to lower pose estimation accuracy, but also allows the FPGA to support 20 raster cores in the Monte-Carlo sampling design of [11] rather than 10 cores in our design. However, as shown in our results above, our optimized design still achieves impressive runtimes and the tradeoff of less parallelism for more accuracy may be particularly appropriate when using this technique for robot scene perception in adversarial environments.



(a) Occluded Scene (b) Dark Scene

Figure 7: Pose estimation accuracy comparison of GRIP [2] and our FPGA implemenations at various distance thresholds in scenes with (a) occlusions, and (b) limited lighting.

Table 2: Resource utilization of a single raster-core design compared to [11].

Implementation	BRAM	DSP	FF	LUT
MC_{FPGA} [11]	24	46	8639	7300
$GRIP_{FPGA}$ (ours)	28	56	11585	10315
% change	+16.7%	+21%	+34.1%	+41.3%

Table 3: Average power and energy results.

Implementation	Power	Energy
$Titan_{all-GPU}$	109 W	11.8 J
$Jetson_{all-GPU}$	3.3 W	6.7 J
FPGA (ours)	**3.85 W**	**0.291 J**

5.5 Power Consumption

Power and energy comparisons among full-GPU and FPGA implementations are shown in Table 3. The FPGA power is collected from the Vivado power analyzer, Titan power is measured through the Nvidia Management Library, and Jetson power numbers are measured through an on-board power monitor. Compared to the all-GPU implementation running on the Titan Xp, our FPGA implementation achieves a 28X improvement in power and 40X improvement in energy. Compared to the low power Jetson, our FPGA implementation dissipates approximately the same amount of power; however, because of the significant runtime advantage (as reported in Table 1) our design achieves a 23X improvement in energy.

6 CONCLUSIONS

Robots have seen widespread proliferation in society. Many of these robots rely on robust scene perception so that they may operate in varied and complex environments. This has led to the proposed use of generative-discriminative algorithms, which combine inference by deep learning with sampling and probabilistic inference models to achieve robust and adaptive perception in adversarial environments. While these algorithms obtain impressive pose estimation accuracy, their computational complexity makes real-time execution a challenge, even if run on a high-end GPU platform. In addition, the computational resources and energy requirements can be quite substantial, which is especially problematic for mobile robots.

In this paper, we have described a novel hardware implementation of a Monte-Carlo sampling algorithm that implements rendering, feature extraction, and inlier comparison in parallel on specialized cores. With our FPGA implementation, we are able to achieve real-time performance without sacrificing accuracy and with significantly reduced energy consumption. In particular, our design runs 30% faster than a high-end GPU implementation with only 2% of the energy consumption, and 95% faster than a low-power GPU implementation, dissipating approximately the same amount of power but with only 4% of the energy consumption. Future work will consider other accuracy/runtime/energy trade-offs for improved robot operation. We will also consider hardware acceleration of other generative algorithms to be combined with discriminative techniques.

REFERENCES

[1] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLOS ONE* 10, 7 (07 2015), 1–46. https://doi.org/10.1371/journal.pone.0130140

[2] X. Chen, R. Chen, Z. Sui, Z. Ye, Y. Liu, R. I. Bahar, and O. C. Jenkins. 2019. GRIP: Generative Robust Inference and Perception for Semantic Robot Manipulation in Adversarial Environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3988–3995.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[4] Marcus Gualtieri, Andreas ten Pas, and Robert Platt. 2017. Category Level Pick and Place Using Deep Reinforcement Learning. *ArXiv* abs/1707.05615 (2017).

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[6] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv:1602.07360* (2016).

[7] Malvin H. Kalos and Paula A. Whitlock. 1986. *Monte Carlo Methods. Vol. 1: Basics*. Wiley-Interscience, USA.

[8] Atsutake Kosuge, Keisuke Yamamoto, Yukinori Akamine, Taizo Yamawaki, and Takashi Oshima. 2019. A 4.8x Faster FPGA-Based Iterative Closest Point Accelerator for Object Pose Estimation of Picking Robot Applications. In *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 331–331. doi:10.1109/FCCM.2019.00072.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[10] J. E. Laird, K. Gluck, J. Anderson, K. D. Forbus, O. C. Jenkins, C. Lebiere, D. Salvucci, M. Scheutz, A. Thomaz, G. Trafton, R. E. Wray, S. Mohan, and J. R. Kirk. 2017. Interactive Task Learning. *IEEE Intelligent Systems* 32, 4 (2017), 6–21.

[11] Y. Liu, G. Calderoni, and R. I. Bahar. 2020. Hardware Acceleration of Monte-Carlo Sampling for Energy Efficient Robust Robot Manipulation. In *IEEE/ACM International Conference on Field Programmable Logic and Applications (FPL)*.

[12] Y. Liu, A. Costantini, Z. Sui, Z. Ye, S. Lu, O. C. Jenkins, and R. I. Bahar. 2018. Robust Object Estimation using Generative-Discriminative Inference for Secure Robotics Applications. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[13] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. 2019. *Layer-Wise Relevance Propagation: An Overview*. Springer International Publishing, Cham, 193–209. https://doi.org/10.1007/978-3-030-28954-6_10

[14] Riku Murai, Paul Kelly, Sajad Saeedi, and Andrew Davison. 2019. Visual Odometry Using a Focal-plane Sensor-processor. (2019). https://www.semanticscholar.org/paper/Visual-Odometry-Using-a-Focal-plane-Murai-Saeedi/934c8ce97b0415fee670d5de9b2d15efc76b8cdc.

[15] Juan Pineda. 1988. A Parallel Algorithm for Polygon Rasterization. In *In Proceedings of Siggraph '88*. 17–20.

[16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).

[17] Michael Schaeferling, Ulrich Hornung, and Gundolf Kiefer. 2012. Object recognition and pose estimation on embedded hardware: SURF-based system designs accelerated by FPGA logic. *International Journal of Reconfigurable Computing* 2012 (2012), 6. doi:10.1155/2012/368351.

[18] Fynn Schwiegelshohn, Eugen Ossovski, and Michael Hübner. 2015. A Fully Parallel Particle Filter Architecture for FPGAs. In *Applied Reconfigurable Computing*, Kentaro Sano, Dimitrios Soudris, Michael Hübner, and Pedro C. Diniz (Eds.). Springer International Publishing, Cham, 91–102.

[19] László Schäffer, Zoltán Kincses, and Szilveszter Pletl. 2018. A Real-Time Pose Estimation Algorithm Based on FPGA and Sensor Fusion. In *International Symposium on Intelligent Systems and Informatics (SISY)*. doi:10.1109/SISY.2018.8524610.

[20] B. G. Sileshi, J. Oliver, and C. Ferrer. 2016. Accelerating Particle Filter on FPGA. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 591–594. doi:10.1109/ISVLSI.2016.66.

[21] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556* (2014).

[22] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. 2017. SmoothGrad: removing noise by adding noise. *ArXiv* abs/1706.03825 (2017).

[23] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 3645–3650. https://doi.org/10.18653/v1/P19-1355

[24] Zhiqiang Sui, Zhefan Ye, and Odest Chadwicke Jenkins. 2018. Never Mind the Bounding Boxes, Here's the SAND Filters. *arXiv:1808.04969* (2018).

[25] Z. Sui, Z. Zhou, Z. Zeng, and O. C. Jenkins. 2017. SUM: Sequential scene understanding and manipulation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3281–3288.

[26] Andreas ten Pas and Robert Platt. 2014. Localizing Handle-Like Grasp Affordances in 3D Point Clouds. In *ISER*.

[27] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. 2018. Deep object pose estimation for semantic robotic grasping of household objects. *arXiv:1809.10790* (2018).

[28] J. Varley, J. Weisz, J. Weiss, and P. Allen. 2015. Generating multi-fingered robotic grasps via deep learning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 4415–4420.

[29] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. 2018. PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes. *Robotics: Science and Systems XIV* (2018). doi:10.15607/RSS.2018.XIV.019.

[30] Luisa M. Zintgraf, Taco Cohen, Tameem Adel, and Max Welling. 2017. Visualizing Deep Neural Network Decisions: Prediction Difference Analysis. In *International Conference on Learning Representations (ICLR)*.