A Reconfigurable Hardware Library for Robot Scene Perception

(Invited Paper)

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

Anthony Opipari
University of Michigan
Dept. of Computer Science
Ann Arbor, MI, USA
topipari@umich.edu

Odest Chadwicke Jenkins
University of Michigan
Dept. of Robotics
Ann Arbor, MI, USA
ocj@umich.edu

R. Iris Bahar
Colorado School of Mines
Dept. of Computer Science
Golden, CO, USA
ribahar@mines.edu

ABSTRACT

Perceiving the position and orientation of objects (i.e., pose estimation) is a crucial prerequisite for robots acting within their natural environment. We present a hardware acceleration approach to enable real-time and energy efficient articulated pose estimation for robots operating in unstructured environments. Our hardware accelerator implements Nonparametric Belief Propagation (NBP) to infer the belief distribution of articulated object poses. Our approach is on average, 26X more energy efficient than a high-end GPU and 11X faster than an embedded low-power GPU implementation. Moreover, we present a *Monte-Carlo Perception Library* generated from high-level synthesis to enable reconfigurable hardware designs on FPGA fabrics that are better tuned to user-specified scene, resource, and performance constraints.

CCS CONCEPTS

• **Hardware → Hardware accelerators**; • **Computer systems organization → Real-time system architecture**; *Robotics*; • **Mathematics of computing → Probabilistic reasoning algorithms**.

KEYWORDS

robotics, hardware acceleration, belief propagation, energy-efficient

ACM Reference Format:
Yanqi Liu, Anthony Opipari, Odest Chadwicke Jenkins, and R. Iris Bahar. . A Reconfigurable Hardware Library for Robot Scene Perception: (Invited Paper). In . ACM, New York, NY, USA, 9 pages.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3561110

1 INTRODUCTION

Determining the position and orientation of an object (i.e., pose estimation) is crucial for robotic perception systems. While neural networks are considered the state-of-art techniques for object pose estimation, they require significant hardware resources and manual labeling efforts for training. Moreover, general purpose GPUs used for neural network inference are usually extremely power hungry and can be challenging to implement on mobile robot platforms with limited power constraints and battery life. Finally, even with all the compute and training efforts, these neural network approaches commonly perform poorly when faced with previously unseen scenarios. While retraining the neural network with updated input data can improve accuracy, this takes time and often the training data for these new scenarios are not available.

Sampling based methods [4, 9, 11, 12, 16] are able to combine the advantages of neural network approaches with the added robustness of probabilistic inference and have been shown to outperform end-to-end neural network approaches when encountering new and challenging environments [2]. However, these sampling methods use iterative processing and have computation patterns that do not map well to the fine-grained parallel execution of GPUs, resulting in slow runtimes and high energy consumption on a GPU.

Recent works have explored custom hardware acceleration of sampling based pose estimation algorithms through FPGAs [8, 10], which has resulted in significant improvements in both runtime and energy consumption. However, these prior works have two main limitations: *i*) they consider pose estimation assuming rigid-body objects, and *ii*) the hardware implementation is fixed to a specific algorithm and cannot be reconfigured to better match design or object constraints for a specific scenario. In this work, we address both these concerns.

Oftentimes, there is an imbalance between software and hardware development effort. While it might take minimal effort to modify the software, redesigning the hardware may take more time and effort and require hardware domain knowledge. With the reconfigurability of FPGAs and the use of High Level Synthesis (HLS), we can reduce the redesign effort by creating an open-source software library that can be used to synthesize new hardware designs[1].

[1]The repository is available at: https://progress.eecs.umich.edu/projects/bp-accelerate/

By creating a software template library, we further help to increase the accessibility for algorithm developers to customize and create accelerators based on their design constraints.

In this paper, we present a novel hardware implementation of belief propagation for articulated object pose estimation and utilize the reconfigurable feature of FPGAs to create a software template library that can be configured to generate hardware designs for implementing pose estimation for both rigid-body and articulated objects.

The rest of this paper is organized as follows. Section 2 summarizes related work in scene perception and hardware acceleration techniques for robotics. Section 3 describes the non-parametric belief propagation algorithm to perform pose estimation. In Section 4 we describe in detail the hardware architectures of the accelerator design and introduce the design of a software library for reconfigurable hardware implementations. Section 5 reports on our experimental results for our FPGA design and compares them to both a high performance and low-power GPU implementation. Finally, Section 6 summarizes our work and discusses important considerations for future open source hardware library development.

2 BACKGROUND

Robot perception is critical for robot manipulation. This step can be computationally expensive and slow; however, there are few works that consider optimizations with hardware acceleration. Of these works, Schaeferling et al. [14] implement a CPU-FPGA solution for SIFT feature based pose estimation. The work of Schäffer et al. [15] uses GPU and IMU sensors for vehicle localization and pose estimation, which is not applicable for 6 degree-of-freedom (DoF) pose estimation. Finally, Kosuge et al. [7] provides an FPGA accelerated iterative closest point algorithm, which cannot produce accurate pose results in unstructured environments with occlusions (as shown in [17]). Finally, recent works [8], [10] proposed an FPGA design for accelerating particle filtering for 6 DoF rigid body object pose estimation. Compared to rigid body object pose estimation, articulated 6 DoF object pose estimation presents additional perception challenges. The configuration space of articulated objects grows in dimensionality with the number of individual parts, which presents a computational challenge for any low-power real-time computing platforms.

Hardware accelerator design often requires significant domain expertise. As a results, there has been recent interest in developing open-source hardware acceleration libraries. Several works use high-level synthesis tools to aid with library development. Kalms et al. [6] and Gorgon and Tadeusiewicz [5] use HLS tools to create an open-source library for image processing. The Xilinx Vitis library [1] provides a set of domain-specific accelerator libraries that can handle applications from image processing to databases to neural networks inference. We take a similar approach of using HLS tools to develop a novel hardware library for object pose estimation, which fills the need for a domain-specific acceleration library in robotic pose estimation. Our FPGA libraries are parameterized and can be configured to target different hardware resource constraints and algorithm runtime and accuracy requirements.

3 ALGORITHM

The goal for sampling-based object pose estimation is to infer the probability distribution of the object pose using observed sensor information. We use belief propagation [13] to efficiently model and compute the distribution for both rigid-body and articulated objects. For articulated objects specifically, we find the pose of each object part separately. In continuous domains such as 6 DoF pose estimation, belief propagation becomes intractable due to its use of integrals. To enable performance in continuous spaces, algorithmic approximations such as pull message passing for nonparametric belief propagation (PMPNBP) [3] have been proposed.

The PMPNBP algorithm uses a Markov Random Field (MRF) to model the object. The MRF is defined by an undirected graph $\mathcal{G}$ composed of a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. Each node $s \in \mathcal{V}$ consists of a hidden variable, X_s, representing the object pose or a part of the object pose and Y_s represents a corresponding observed data related to the part. Denoting all hidden variables in the graph as X and all observed variables as Y, the joint probability distribution of the random variables is given by the factorization:

$$P(X, Y) \propto \prod_{(s,t)\in\mathcal{E}} \psi_{s,t}\left(X_s, X_t\right) \prod_{s\in\mathcal{V}} \phi_s\left(X_s, Y_s\right), \tag{1}$$

where $\phi_s(X_s, Y_s)$ is the unary likelihood for node s, which describes the correspondence of an object part's pose with its observation. The function $\psi_{s,t}(X_s, X_t)$ denotes the pairwise likelihood between neighboring nodes, which measures how compatible part s at pose X_s is with respect to part t at pose X_t based on the pair's articulation constraints.

For 6 DoF pose estimation, the observation is the RGBD data from the camera sensor. The Markov random field is generated using a Unified Robot Description Format (URDF) that describes the joint limits between articulated object parts (articulation constraints), and the 3D mesh models of each object part. We compare the rendered mesh model depth to the observation depth as an estimation of unary likelihood and use the URDF file to create joint constraints as needed for pairwise likelihood calculations. Details of the unary likelihood calculation are given in Section 4.2.

3.1 Message passing

The PMPNBP algorithm defines a particle-based message passing approach for approximate inference of each node's belief distribution. The message passing scheme is illustrated in Fig. 1 with an articulated clamp object, where each node X_s corresponds to the pose of a part of the object and messages are passed between neighboring nodes. The belief distribution *bel* of each node and messages m between nodes are each represented by a set of weighted particles $\{w, \mu\}_{i=1}^{M}$, where μ represents a sampled pose hypothesis for one object part, and w is the corresponding sample's weight. In the **message passing stage**, a message $m_{ts}\left(X_s\right)$ from node t to node s constitutes the distribution of the belief of node s informed by its neighboring node t. This is mathematically expressed as:

$$m_{ts}^{n}\left(X_s\right) = \sum_{X_t\in\mathbb{X}_t} \phi_t(X_t, Y_t)\psi_{s,t}(X_s, X_t) \prod_{u\in\rho(t)\setminus s} m_{ut}^{n-1}(X_t), \tag{2}$$

where $\rho(t) \setminus s$ denotes neighboring nodes of t excluding node s, and $\mathbb{X}_t$ denotes the particle set of node t. To compute $m_{ts}^{n}(X_s)$ at iteration n, first $\mu_{ts}^{(i)}$ is sampled from the belief $bel_s^{n-1}(X_s)$. Next, these samples are passed to the neighboring nodes of t where the weights $\{w_{ts}^{(i)}\}_{i=1}^{M}$ are computed. This message update process in outlined in Algorithm 1.

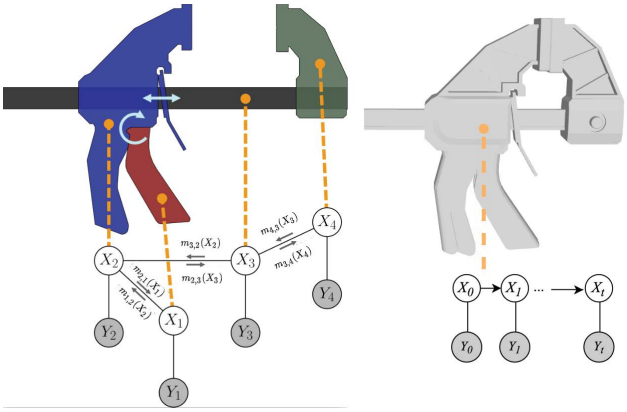

Figure 1: *left:* A clamp represented as an articulate object by an MRF. Each hidden node X_s corresponds to the pose of a part of the object with the corresponding observed node Y_s. Messages are passed bi-directionally between nodes (X_s, X_t) connected by an edge. *right:* A clamp represented as rigid body by a Hidden Markov Model.

Algorithm 1: PMPNBP Message Update [3]

Input : Incoming neighbor messages
$m_{ut}^{n-1}(X_t) = \{(\mu_{ut}^{(i)}, w_{ut}^{(i)})\}_{i=1}^{M} \ \forall u \in \rho(t) \setminus s$
Output: Updated outgoing message
$m_{ts}^{n}(X_s) = \{(\mu_{ts}^{(i)}, w_{ts}^{(i)})\}_{i=1}^{M}$
Function `message_update`(m_{ut}^{n-1}):
  Sample M outgoing particles $\{\mu_{ts}^{(i)}\}_{i=1}^{M}$ from $bel_s^{n-1}(X_s)$
  foreach $\mu_{ts}^{(i)}$ **do**
    sample $\hat{X}_t^{(i)} \sim \psi_{s,t}(X_s = \mu_{ts}^{(i)}, X_t)$ ▷ pairwise sampler
    $w_{\text{unary}}^{(i)} \leftarrow \phi_t(X_t = \hat{X}_t^{(i)}, Y_t)$
    $w_{\text{neigh}}^{(i)} \leftarrow$
    $\prod_{u \in \rho(t)\setminus s} \left(\sum_{j=1}^{M} w_{ut}^{(j)} \psi_{s,t}(X_s = \mu_{ts}^{(i)}, X_t = \mu_{ut}^{(j)}) \right)$
    $w_{ts}^{(i)} \leftarrow w_{\text{unary}}^{(i)} \cdot w_{\text{neigh}}^{(i)}$
  end
  $m_{ts}^{n}(X_s) \leftarrow \{(\mu_{ts}^{(i)}, w_{ts}^{(i)})\}_{i=1}^{M}$
  return $m_{ts}^{n}(X_s)$

3.2 Belief update

In the **belief update stage**, the marginal belief of the hidden variable of node s is expressed as the product of all incoming messages from neighboring nodes $t \in \rho(s)$ weighted by the node's unary likelihood, as shown in Equation 3:

$$bel_s^n(X_s) \propto \phi_s(X_s, Y_s) \prod_{t \in \rho(s)} m_{ts}^n(X_s). \tag{3}$$

The node s receives all weighted incoming messages from the neighboring nodes and combines them into a set $\{(\mu_s^{(i)}, w_s^{(i)})\}_{i=1}^{T}$. The particles are then normalized and resampled as a new set of belief particles using importance sampling for the next iteration. Pseudocode for the belief update stage is included in Algorithm 2.

Algorithm 2: PMPNBP Belief Update [3]

Input : Incoming neighbor messages
$m_{ts}^{n}(X_s) = \{(\mu_{ts}^{(i)}, w_{ts}^{(i)})\}_{i=1}^{M} \ \forall t \in \rho(s)$
Output: Updated belief $bel_s^n(X_s) = \{\mu_s^{(i)}\}_{i=1}^{M}$
Function `belief_update`$(m_{ts}^{n}(X_s))$:
  foreach $t \in \rho(s)$ **do**
    foreach $\mu_{ts}^{(i)}$ **do**
      $w_{ts}^{(i)} \leftarrow w_{ts}^{(i)} \cdot \phi_s(X_s = \mu_{ts}^{(i)}, Y_s)$
      Normalize weights $\{w_{ts}^{(i)}\}_{i=1}^{M}$, resulting in $\sum_{i=1}^{M} w_{ts}^{(i)} = 1$
    end
  end
  Combine all T of the incoming message particles into a set $\{(\mu_s^{(i)}, w_s^{(i)})\}_{i=1}^{T}$
  Normalize the weights in this combined particle set, resulting in $\sum_{i=1}^{T} w_s^{(i)} = 1$.
  Perform resampling to sample new set of $\{\mu_s^{(i)}\}_{i=1}^{M}$ that represents bel_s^n
  return $bel_s^n = \{\mu_s^{(i)}\}_{i=1}^{M}$

Algorithm 3: Pull Message Passing for Nonparametric Belief Propagation (PMPNBP) [3]

foreach *iteration* $n \in [1, N]$ **do**
  foreach *edge* $(s,t) \in \mathcal{E}$ **do**
    $m_{ts}^{n}(X_s) \leftarrow$ `message_update`$(m_{ut}^{n-1}(X_t) \ \forall u \in \rho(t) \setminus s)$
    $m_{st}^{n}(X_t) \leftarrow$ `message_update`$(m_{us}^{n-1}(X_s) \ \forall u \in \rho(s) \setminus t)$
  end
  foreach *node* $s \in \mathcal{V}$ **do**
    $bel_s^n(X_s) \leftarrow$ `belief_update`$(m_{ts}^{n}(X_s) \ \forall t \in \rho(s))$
  end
end

The belief update and message update stages are performed alternatively and iteratively until the belief at each node converges. Collectively, these processing stages makes up the pull message passing algorithm, as is outlined in Algorithm 3.

3.3 Particle filtering

The particle filtering algorithm [18, 19] performs inference on a Hidden Markov Model (HMM), which can be represented as a special case for Non-parametric belief propagation. Such representation can be used to model the probabilistic distribution any rigid-body objects where we do not expect the changes in the object geometry as we seen in Fig. 1. With the sequential nature of the HMM, we can interpret particle filter using Belief Propagation(BP) to perform *forward* message passing from node x_{t-1} to node x_t. The message passing from Eq. 2 can be expressed as:

$$m_{t,t+1}(x_{t+1}) \propto \int p(x_{t+1}|x_t) p(y_t|x_t) m_{t-1,t}(x_t) dx_t \tag{4}$$

At the starting point, the forward message can be expressed as:

$$m_{0,1}(x_1) \propto \int p(x_1|x_0)p(x_0)p(y_0|x_0)dx_0 \propto p(x_1|y_0) \quad (5)$$

We see a forward message is proportional to the conditional probability of given state conditioned on all previous observation. By applying the observation at the current state, we can get the posterior probability of the current state Eq. 7.

$$m_{t-1,t}(x_t) \propto p(x_t|y_0, y_1, ...y_{t-1}) \quad (6)$$

$$m_{t-1,t}(x_t)p(y_t|x_t) \propto p(x_t, |y_0, y_1, ...y_t) = bel(x_t) \quad (7)$$

Since computing the integral over non-linear and high-dimensional state space is intractable, we can use Monte-Carlo methods to approximate the message with set of weighted particles $m_{t-1,t}(x_t) = \{(\mu_t^{(i)}, w_t^{(i)})\}_{i=1}^{M}$ Weights for each sample w_t are calculated by the unary potential of sample x_t and the importance sample is used to generate a new sample set with their corresponding weights.

4 IMPLEMENTATION

Our hardware implementation takes inputs in the form of a user-defined graphical model, $\mathcal{G}$, and a set of randomly initialized belief and message particles and performs belief propagation using the pull message passing algorithm described in Section 3 until all particles converge. Our design centers around three main optimizations in hardware: *i*) overlapping the belief update and message update stages using deep pipelining, *ii*) customizing hardware computing units for task-specific computing, and *iii*) implementing a flexible hardware library for easy reconfiguration. Finally, all computations are done using mixed-precision fixed point arithmetic to improve speed efficiency on the FPGA (where object pose is represented using 16-bit fixed-point and depth is 12-bit fixed point). We now highlight these optimizations in more detail.

4.1 Overlapping Belief and Message Updates

The biggest improvement with a custom hardware design is obtained by overlapping the computation of the belief update with the message update. The dataflow for one iteration of belief update and message update is shown in Fig. 2. For all samples in the message samples set $\{\mu_{ts}\}_{i=1}^{M}$ on directed edge (t, s), we create three concurrent datastreams to overlap the belief update and message update computations. *Datastream0* contains the belief unary unit, which is in charge of calculating the unary likelihood $\phi(X_s = \mu_{ts}, Y_s)$ for the belief update stage on message sample μ_{ts}. *Datastream1* goes through the pairwise sampler for drawing a compatible sample $\hat{X}_t$ from $\psi_{s,t}(X_s = \mu_{ts}, X_t)$ and then calculates the unary likelihood for the compatible sample $\hat{X}_t$ as message unary. *Datastream2* fetches the message sample set from the neighboring edge (u, t), calculates the pairwise point-wise potential to the message sample μ_{st} and accumulates the result as w_{neigh} for message update. For the pairwise potential functions $\psi_{s,t}(X_s = \mu_{ts}^{(i)}, X_t = \mu_{ut}^{(j)})$, if particle X_s falls within the joint limits of s with respect to t, then the pairwise likelihood is assigned a value of 1. Otherwise, the potential is the exponential of the negative error between X_s and the nearest joint limit. Within the pairwise point-wise unit, we further create parallel computation units for $\mu_{ut}^{(j)}$ to process all M samples for all t nodes neighboring node u. By using datastreams, we can avoid saving intermediate values for each stage of computation to reduce memory accesses, which results in energy saving. Each stage in the dataflow is pipelined such that the three datastreams can be computationally balanced and there is no stalling before the resampler.

4.2 Raster Core

For belief propagation used in the 6 DoF pose estimation problem, the unary likelihood is estimated by comparing the mesh model of part s rendered at pose X_s with the observation Y_s. This unary likelihood can be expressed as

$$\phi_s(X_s, Y_s) = \phi_s^{rgb}(X_s, Y_s^{rgb})\phi_s^{depth}(X_s, Y_s^{depth}), \quad (8)$$

where ϕ^{rgb} is evaluated by the percentage of pixel intersection between rendered object mask and the segmentation map, which can be generated by a segmentation neural network. The variable ϕ^{depth} is estimated by the pixel distance error between the rendered depth map to the observation depth map Y_s^{depth}. If the distance error is below a certain threshold, we count the pixel as an inlier.

The unary likelihood computation requires rasterization, which converts a mesh model defined by vertices and faces to a raster depth image. The result of rasterization is a depth image of what a 3D object would look like at a certain pose. To implement unary likelihood in hardware, we use a specialized *raster core* processing unit. Similar to the one described in [8], the raster core pipelines the 3D transformation with rasterization and inlier comparison steps for a given sample. An example is shown in Fig. 3. During the rasterization process, the rasterized pixel is immediately compared with the corresponding observation pixel to check if the pixel is an inlier, without waiting for the entire object to be rasterized first. Compared to a GPU implementation, which is optimized for graphical rendering, our *raster core* takes advantage of pipelining and overlaps the process of rendering and inlier comparison, which has to be performed sequentially on a GPU. The raster core unit is modularized and can be replicated to process multiple samples concurrently to further improve its throughput.

4.3 Parallel pairwise potential core

For the 6 DoF pose estimation problem, the sample pose and joint limit are represented using dual quaternions. For two neighboring nodes s and t, the joint limit between the nodes s and t is defined by $R_{t|s} = [dq_{t|s}^{min}, dq_{t|s}^{max}]$. For two samples μ_{ts} and μ_{ut}, the pairwise potential is calculated by the distance of μ_{ts} to the joint constraints at μ_{ut}: $[dq_{t|s}^{min} * \mu_{ut}, dq_{t|s}^{max} * \mu_{ut}]$. If μ_{ts} falls within this range, the pairwise potential score is 1, otherwise the pairwise potential score is exponentially decreasing with the distance from the range limit.

For our FPGA implementation, we partitioned our message sample buffer according to the number of parallel pairwise potential cores. Therefore, each pairwise potential core can fetch from the buffer concurrently and the outputs are accumulated in an adder tree, as shown in Fig. 4. Pairwise potential cores are implemented with DSPs and lookup tables to compute the dual quaternion transformation and comparison. Note that fetching data, pairwise potential computation, and adder tree computation steps are pipelined, which increases throughput when computing pairwise potential.

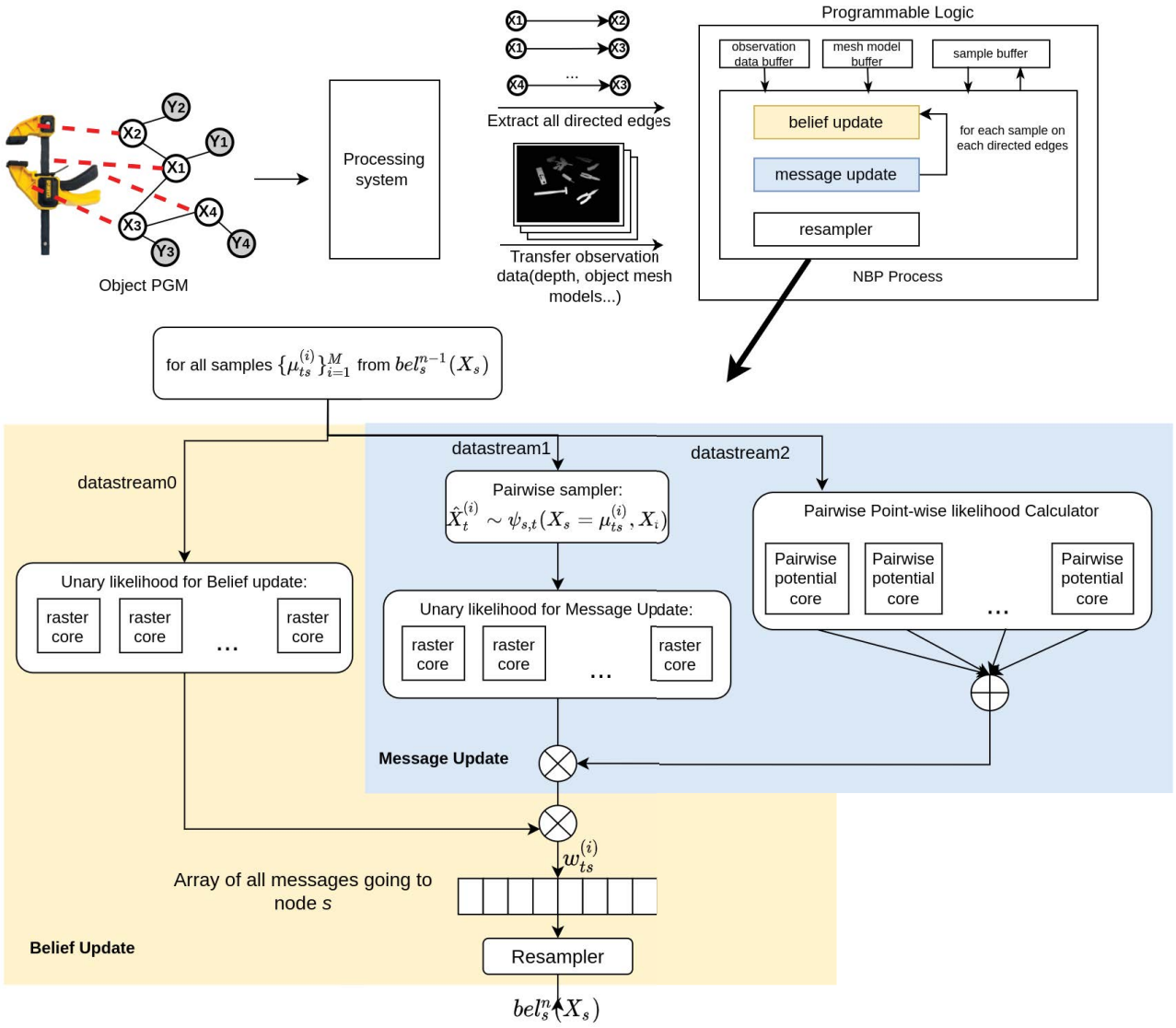

Figure 2: The hardware system for our nonparametric belief propagation (NBP) algorithm. The processing system (CPU) takes the user input object PGM, extracts all directed edges, and transfers the observation data to the programmable logic through the AXI-4 protocol. Using programmable logic, we implement the NBP algorithm by creating three parallel dataflows, overlapping the belief update with message update computation. Within the unary likelihood and pairwise likelihood unit, we further create parallel processing units to increase throughput. We save the message weight w_{ts} in an array and the resampler performs importance sampling to generate a new set of beliefs for node s for the next iteration.

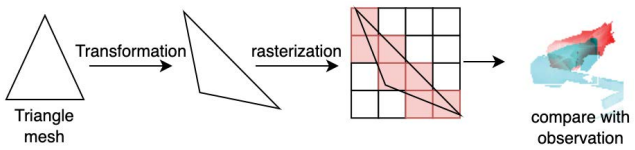

Figure 3: Example of the raster core processing each triangle of the object mesh model in sequence. On the right, the red area is the rastered depth image and the blue area is the observation from the depth camera.

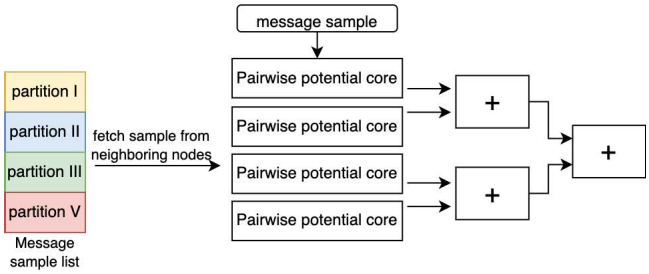

Figure 4: Example of 4 parallel pairwise potential units

4.4 Reconfigurability

In Section 3, we explained that particle filtering is a special case of nonparametric belief propagation (NBP). Therefore, we can create a uniform HLS-based library that can generate accelerator designs with different data flows and streams to satisfy either of the algorithms' requirements. The architecture flow of our *Monte-Carlo Perception Library* is shown in Fig. 5. In our library design, particle filtering and NBP share common hardware modules including the belief unary module, the weight calculator/merger, and the resampling module, while the NBP algorithm has a additional hardware component — a pairwise likelihood unit for calculating the messages from the neighboring nodes. The library is able to reroute data streams according to user configurations. For such a library, we can handle three types of pose estimation algorithms: particle filtering for rigid-body objects, multiple particle filters for articulated object parts, and NBP for articulated object pose estimation.

To give more control of the accelerator design to users without requiring detailed hardware knowledge, we created a user interface to provide some knobs for users to adjust to generate specific designs tuned for their applications. We allow users to configure the accelerator based on different resource allowances, as well as power and energy constraints, as shown in Fig. 6. The input to the interface is a Unified Robot Description File (URDF) for the test object and the user parameters, which are used to create two configuration files. The *synthesize time configuration file* includes a list of algorithms the accelerator is running, the number of parallel hardware components, the size of the memory scratchpad inside the raster core, and algorithm parameters. The file determines the amount of hardware resources needed by the accelerator and is used to generate the desired customized hardware (i.e., an HLS design and IP) mapped onto an FPGA. With a complete accelerator design on an FPGA, the *runtime configuration file* is integrated with the software interface and used to tradeoff runtime and accuracy of the design by varying the number of samples, algorithm iterations, test objects, and test scenes. In this way, the user can tune the system based on application requirements.

5 EXPERIMENTAL RESULTS

We implemented our hardware design using Xilinx Vivado HLS and tested it on a Xilinx FPGA ZCU102 Evaluation Board running at 200 MHz. We compare the performance to a reference GPU implementation running on two different GPU platforms: a high-performance Nvidia Titan Xp running at 1.4GHz, and an embedded low-power platform Nvidia Jetson AGX Xavier running at 1.37GHz. Note that both GPU platforms run at much higher clock frequency with more hardware resources than the FPGA. To measure power on these different platforms, we used the Vivado power analyzer to measure power on the FPGA, the NVIDIA Management Library to estimate power on the Titan Xp, and an on-board power monitor to collect readings for the Jetson AGX. The reference design on GPU utilizes the high degree of parallelism in CUDA kernels and the OpenGL library to render and process pose samples in parallel.

5.1 6 DoF pose estimation for tool dataset

We tested static frame 6 DoF pose estimation using a tool dataset that consists of scenes of hand tools randomly placed on the tabletop, as used in [12]. We tested the following articulated objects,

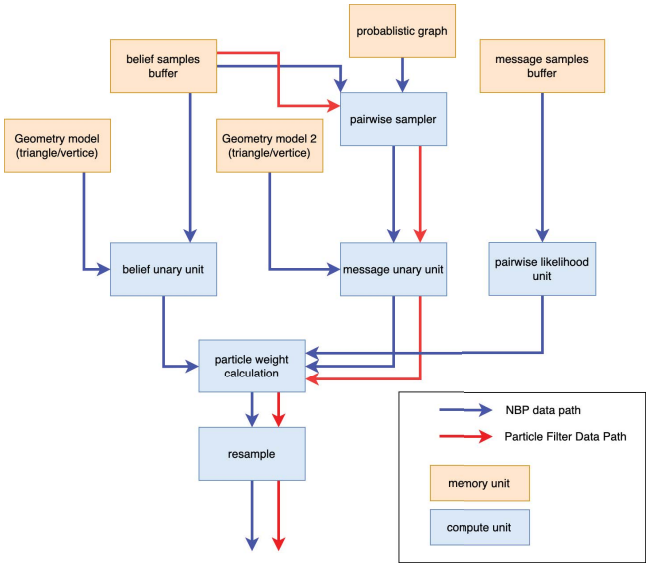

Figure 5: The overall architecture for the generalized library. The Belief Propagation datapath is shown in blue and the datapath that only belongs to particle filter is shown in red. Based on the user specification, the library can automatically select the datapath and hardware components for different algorithms.

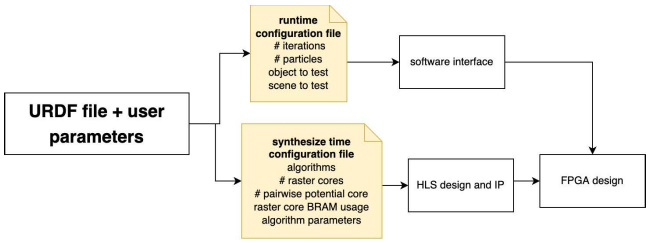

Figure 6: User interface for generalized library.

shown in Table 1: a clamp, 3 types of pliers, a screwdriver, and a flashlight, as they have different mesh sizes and articulations. For the experiment, we evaluated our design with 256 particles for each directed edge and randomly initialized the particles from the image segmentation. Since this experiment was meant focus specifically on the hardware acceleration of the belief propagation algorithm, the image segmentation was generated from the groundtruth, rather than a neural network output. We ran the belief message passing loop for 100 iterations. An example run of the algorithm is shown in Figure 7, where part (a) shows the initial belief and (b) shows the final converged belief.

5.1.1 Runtime. We compare the per-iteration (one pass of message update and belief update) runtime for the different objects on three different platforms. The runtime comparison for each articulated object is shown in Fig. 8. Compared to the GPU implementations, except for the clamp and flashlight, our FPGA design achieves better or comparable results to the Titan Xp and significant runtime improvements compared to the Jetson AGX. Note that running on

(a) (b)

Figure 7: Static frame 6 DoF pose estimation with PMPNBP, (a) represents the initial random sampled belief of the clamp object and (b) shows the convergence of samples after 100 iterations of belief message passing.

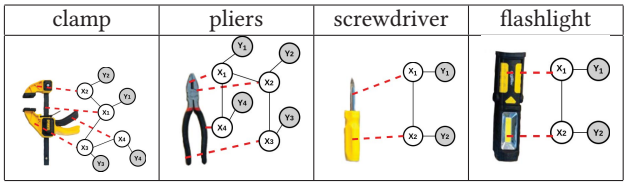

Table 1: Test objects with corresponding PGMs

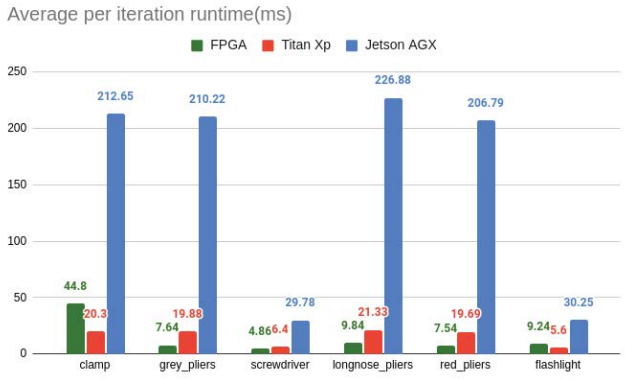

Figure 8: Average per-iteration runtime (in ms) for articulated object 6 DoF pose estimation for tool dataset.

the FPGA, object rendering times vary depending on the specific object's size and mesh size. As often is the case, rendering is the critical path for the iteration runtime. Even though the graph complexity is the same for the clamp and pliers, the clamp has 10-15X more pixels rendered compared to other objects, which results in a longer runtime on the FPGA. However, for the GPU implementation, most of the computing time is spent on the inlier computation and rendering is highly optimized through the OpenGL library with a large number of parallel threads. Our FPGA implementation shows runtime improvements by using optimal dataflow and pipelining the computation while achieving parallelism by creating concurrent processing units. We can further reduce the rendering time on the FPGA by creating more parallel raster cores if we have more hardware resources on board.

To put our runtime improvements in better perspective, we also compare our results to previous works. The CPU implementation of the PMPNBP algorithm in [3] for a simple 2D problem takes over 50 sec per-iteration for 100 particles and the CPU-GPU mixed implementation in [12] on the same tool dataset reports 0.5s–2s per-iteration runtime with 300 particles. We see our GPU and FPGA implementations have greatly accelerated the PMPNBP algorithm compared to these implementations *by 1-2 orders of magnitude.*

5.1.2 Power and Energy. Power and energy comparisons on different hardware platforms are shown in Table 2. The power is measured as an average over 100 iterations of the pull message passing algorithm and energy is measured by multiplying power with average per-iteration runtime. Our FPGA design is more than 25X power efficient and 26X energy efficient compared to the Titan Xp. Compared to the low power Jetson AGX we are 1.4X more power efficient and at least 14X more energy efficient.

	FPGA	Titan Xp	Jetson AGX
power	4.7W	119.85W	6.4W
power saving	-	>25x	>1.4x
energy	65.73 mJ	1747.41mJ	977.66mJ
energy saving	-	>26x	>14x

Table 2: 6 DoF pose estimation power and energy comparison for the tool dataset.

5.2 Reconfigurable library

In our experiments, we also evaluated runtime, power, and energy improvement with different accelerator designs generated by our *Monte-Carlo Perception Library*. By creating our perception library, we can easily experiment with different design configurations and allow for easy design space exploration. Here, we experiment with runtime and power impact by adjusting different numbers of computation units. We first create an optimal design for each articulated object as shown in Table 3. For special cases such as screwdrivers and flashlights, the message passing happens only between two nodes and there is no pairwise potential passing from the neighbor of the neighboring nodes. Therefore, those objects can be evaluated with accelerator design with unary units only. By tailoring specific hardware resources for each testing object, we are able to further improve the runtime and power dissipation of the algorithm compared to the previous fixed design shown in Fig. 8. The improvements in runtime, power and energy in shown in Figures 9 10 11. In particular, we see that through optimal configurations for individual object, we are able to further reduce energy cost by another 20%–30%.

We also used our perception library to evaluate different particle filtering-based pose estimation algorithms. We create two experiments, one treating the object as a rigid body object and the other allowing for articulations in the objects using part-based particle filtering (Part-based PF), described in [12]. In Table 4 we showed the runtime and power for two particle filtering experiments.

Finally, we performed design space exploration with the library design itself. Design space exploration allows users to find the "sweet-spot" of the design under their particular constraints. We created accelerator designs with different configurations and explored the tradeoff between runtime and power, as shown in Fig. 12. The design exploration is performed by using different algorithms

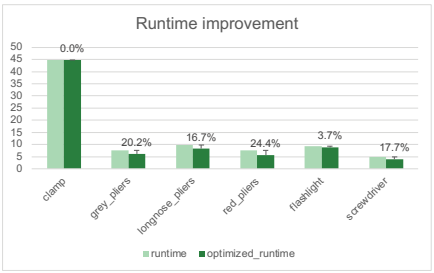

Figure 9: Runtime improvement

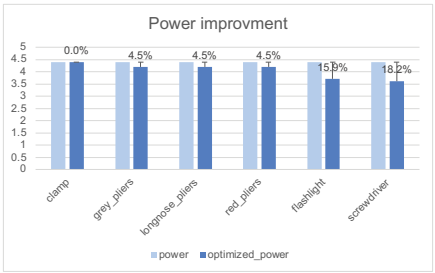

Figure 10: Power improvement

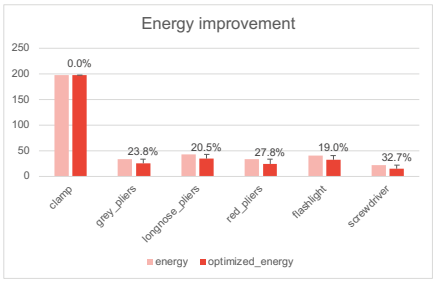

Figure 11: Energy improvement

Object	Optimal Configuration
clamp	**8** raster core per unary unit + **1** pairwise potential core + **150x150** raster core memory
grey pliers	**8** raster core per unary unit + **1** pairwise potential core + **80x80** raster core memory
red pliers	**8** raster core per unary unit + **1** pairwise potential core + **80x80** raster core memory
longnose pliers	**8** raster core per unary unit + **1** pairwise potential core + **80x80** raster core memory
screwdriver	**8** raster core per unary unit + **0** pairwise potential core + **70x70** raster core memory
flashlight	**8** raster core per unary unit + **0** pairwise potential core + **80x80** raster core memory

Table 3: runtime and power for different articulated object pose estimation at their optimal configuration. Since screwdriver and flashlight has only two nodes, the message passing happens only between two nodes so there is no need to calculate the pairwise potential and therefore, they can be evaluated with IP design with unary unit only.

	Part-based PF(16 raster cores)	Rigid object PF(16 raster cores)
power	3.1 W	3.1 W
clamp	15.7ms	8.15ms
grey_pliers	3.4ms	2.05ms
red_pliers	3.4ms	2.1ms
longnose_pliers	6.7ms	3.2ms
screwdriver	3.0ms	1.6ms
flashlight	3.2ms	3.1ms

Table 4: Runtime and power for particle-filtering based pose estimation for rigid body objects and articulated objects

and adjusting different numbers of parallel hardware components, while using the same number of particles for all the designs. We see that for NBP algorithms, increasing the number of pairwise potential units does not help with the overall improvement in runtime given the fact that rasterization is the critical path. In general, increasing the number of raster core units has more benefits in reducing overall energy consumption. NBP algorithms also have much higher energy consumption compared to part-based particle filtering algorithms. As shown in pose estimation accuracy from [12], part-based particle filtering actually has shown decent pose estimating accuracy with symmetric objects (such as screwdrivers). This experiment informs us that we can use part-based particle filtering with symmetric objects as it saves significant energy and runtime.

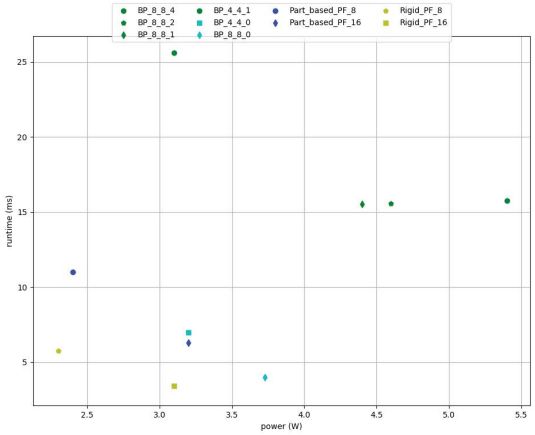

Figure 12: Design space exploration.

6 CONCLUSION

In this paper, we present novel hardware architectures for accelerating sampling-based pose estimation, critical for robotic applications. Our accelerator design achieves runtime and power/energy advantages through hardware customized to address the main bottlenecks of the pose-estimation algorithm. In particular, we use deep pipelining to overlap belief and message update stages, pipeline balancing across different data streams, and elimination of unnecessary computation to optimize the computation. In addition, our flexible *Monte-Carlo Perception Library* allows users to modify the accelerator design based on their performance requirements and hardware constraints, which results in further runtime and energy saving compared to a fixed design. For future work, we hope to extend our perception library to include other machine learning algorithms appropriate for robotics, including those used for motion planning.

ACKNOWLEDGMENT

This work is supported by equipment grants from Nvidia and Xilinx Corporations, and NSF grant #2128036.

REFERENCES

[1] [n.d.]. Xilinx Vitis Library. https://github.com/Xilinx/Vitis_Libraries/tree/master/vision.

[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] Karthik Desingh, Shiyang Lu, Anthony Opipari, and Odest Chadwicke Jenkins. 2019. Efficient nonparametric belief propagation for pose estimation and manipulation of articulated objects. *Science Robotics* 4, 30 (2019), eaaw4523. https://doi.org/10.1126/scirobotics.aaw4523 arXiv:https://www.science.org/doi/pdf/10.1126/scirobotics.aaw4523

[4] Karthik Desingh, Anthony Opipari, and Odest Chadwicke Jenkins. 2018. Pull Message Passing for Nonparametric Belief Propagation. *arXiv:1807.10487 [cs]* (2018). arXiv:1807.10487 http://arxiv.org/abs/1807.10487

[5] Marek Gorgon and Ryszard Tadeusiewicz. 2000. Hardware-based image processing library for Virtex FPGA. In *Reconfigurable Technology: FPGAs for Computing and Applications II*, Vol. 4212. SPIE, 1–10.

[6] Lester Kalms, Ariel Podlubne, and Diana Göhringer. 2019. Hiflipvx: An open source high-level synthesis fpga library for image processing. In *International Symposium on Applied Reconfigurable Computing*. Springer, 149–164.

[7] Atsutake Kosuge, Keisuke Yamamoto, Yukinori Akamine, and Takashi Oshima. 2021. An SoC-FPGA-Based Iterative-Closest-Point Accelerator Enabling Faster Picking Robots. *IEEE Transactions on Industrial Electronics* 68, 4 (2021), 3567–3576. https://doi.org/10.1109/TIE.2020.2978722

[8] 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)*.

[9] Yanqi Liu, Alessandro Costantini, R Bahar, Zhiqiang Sui, Zhefan Ye, Shiyang Lu, and Odest Chadwicke Jenkins. 2018. Robust object estimation using generative-discriminative inference for secure robotics applications. In *Proceedings of the International Conference on Computer-Aided Design*. ACM, 75.

[10] Yanqi Liu, Can Eren Derman, Giuseppe Calderoni, and R. Iris Bahar. 2020. Hardware Acceleration of Robot Scene Perception Algorithms. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–8.

[11] Anthony Opipari, Chao Chen, Shoutian Wang, Jana Pavlasek, Karthik Desingh, and Odest Chadwicke Jenkins. 2021. Differentiable Nonparametric Belief Propagation. *arXiv preprint arXiv:2101.05948* (2021).

[12] Jana Pavlasek, Stanley Lewis, Karthik Desingh, and Odest Chadwicke Jenkins. 2020. Parts-Based Articulated Object Localization in Clutter Using Belief Propagation. *arXiv:2008.02881 [cs]* (2020). arXiv:2008.02881 http://arxiv.org/abs/2008.02881

[13] Judea Pearl. 1982. Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In *Proceedings of the Second AAAI Conference on Artificial Intelligence* (Pittsburgh, Pennsylvania) *(AAAI'82)*. AAAI Press, 133–136.

[14] 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.

[15] 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.

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

[17] Zhiqiang Sui, Zhefan Ye, and Odest Chadwicke Jenkins. 2018. Never Mind the Bounding Boxes, Here's the SAND Filters. *CoRR* abs/1808.04969 (2018). arXiv:1808.04969 http://arxiv.org/abs/1808.04969

[18] Andreas Ten Pas and Robert Platt. 2016. Localizing handle-like grasp affordances in 3d point clouds. In *Experimental Robotics*. Springer, 623–638.

[19] Jeong Woo, Young-Joong Kim, Jeong-on Lee, and Myo-Taeg Lim. 2006. Localization of mobile robot using particle filter. In *2006 SICE-ICASE International Joint Conference*. IEEE, 3031–3034.