2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

PERCH 2.0 : Fast and Accurate GPU-based Perception via Search for Object Pose Estimation

Aditya Agarwal[1], Yupeng Han[1], Maxim Likhachev[1]

***Abstract*— Pose estimation of known objects is fundamental to tasks such as robotic grasping and manipulation. The need for reliable grasping imposes stringent accuracy requirements on pose estimation in cluttered, occluded scenes in dynamic environments. Modern methods employ large sets of training data to learn features in order to find correspondence between 3D models and observed data. However these methods require extensive annotation of ground truth poses. An alternative is to use algorithms that search for the best explanation of the observed scene in a space of possible rendered scenes. A recently developed algorithm, PERCH (PErception Via SeaRCH) does so by using depth data to converge to a globally optimum solution using a search over a specially constructed tree. While PERCH offers strong guarantees on accuracy, the current formulation suffers from low scalability owing to its high runtime. In addition, the sole reliance on depth data for pose estimation restricts the algorithm to scenes where no two objects have the same shape. In this work, we propose PERCH 2.0, a novel perception via search strategy that takes advantage of GPU acceleration and RGB data. We show that our approach can achieve a speedup of 100x over PERCH, as well as better accuracy than the state-of-the-art data-driven approaches on 6-DoF pose estimation without the need for annotating ground truth poses in the training data. Our code and video are available at https://sbpl-cruz.github.io/perception/.**

I. Introduction

For robots to operate successfully in everyday indoor environments they need to be able to interact with objects in a safe and reliable manner. Such interaction requires correct identification of object categories as well as their location and orientation in the 3D world. Variations in objects (color and shape) as well as the environment (lighting conditions, clutter, occlusions) make this a challenging task.

In many instances, 3D models of objects of interest are available and early work in 3D object detection focused on detecting features from these models and matching those to the observed scene. Feature-based methods [1]–[3] typically require rich textures to be present on objects and even when features are present, fail to find good estimates when objects are occluded. Moreover, estimating the pose of each object in isolation may not lead to a globally feasible and optimal solution that fully explains the observed scene [4]. Following the success of convolutional neural networks (CNNs) on computer vision tasks, they have also been extended to estimate object poses in 3D space [5]–[20]. However, these methods require large sets of training data to be able to estimate poses accurately. The required dataset of poses scales poorly with the number of objects since networks need to be trained with images from as many viewpoints as possible and with varying degrees of inter-object occlusions to avoid overfitting. Moreover, the task of annotating poses is non-trivial and requires specialized tools [21] unlike annotation for tasks such as 2D object detection and instance segmentation which can be easily crowd-sourced.

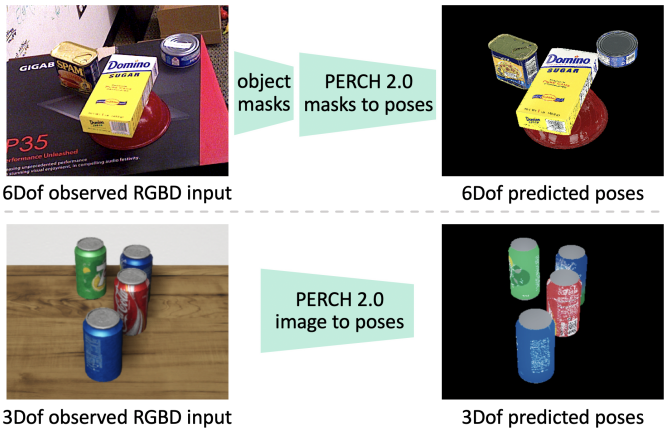

Fig. 1: *Top*: PERCH 2.0 pipeline for 6-Dof Pose estimation *Bottom*: PERCH 2.0 pipeline for 3-Dof pose estimation

Methods that rely on synthesizing scenes and matching these with observed scenes [22]–[26] overcome shortcoming of feature and learning based methods but tend to be slow. Specifically, PERCH [24]–[26] is a recent work that introduces a global matching objective function and does such a search in an efficient manner. While learning-based methods have been beneficiaries of advancements in GPU hardware and availability of compatible software computing platforms like CUDA [27], methods such as PERCH have so far remained restricted to CPU. In this work we propose PERCH 2.0, a perception via search technique that remedies this shortcoming and offers an order of magnitude reduction in runtime. Our contributions are mainly the following :

- A fully parallel GPU-based search formulation to achieve significant speedup over PERCH for 3-Dof pose estimation
- Incorporation of RGB sensor data into the objective function used by PERCH 2.0, allowing the algorithm to handle scenarios where depth data alone is not sufficient to estimate the 3-Dof poses
- A PERCH 2.0 based discriminative-generative framework for 6-Dof pose estimation that eliminates the need for ground truth pose annotation in the training data and outperforms state-of-the-art purely discriminative approaches

[1]The Robotics Institute, Carnegie Mellon University, PA, USA {adityaa2,yupengh,mlikhach}@andrew.cmu.edu

II. Related Work

A. Discriminative Approaches

Discriminative approaches traditionally used hand-crafted local 3D features to establish 2D to 3D correspondences between the observed image and the 3D model and recover the object pose [1]–[3]. Other traditional approaches computed similarity scores over regions of observed images with an object template (obtained by rendering 3D models) to obtain the best match and corresponding pose [28]–[30]. However recent advancements in deep learning has led to 2D object detectors being extended for the task of 6-Dof pose estimation [5]–[20]. Of these, some regress directly to pose estimates [13], tying the pose estimation to camera intrinsics and thus introducing errors if the camera is changed. Others localize object keypoints in image space [6], [7], [10], [12], [14], [15], [19] which often results in ambiguities for objects with symmetries or requires explicit handling of symmetries. Others score discretized poses [17], [20] which is independent of camera parameters and object symmetries. However methods in each of these categories require extensive annotation of ground truth 6-Dof poses in the training data. Recent works [31], [32] have proposed to counter this through synthetic data but these methods still need to be trained for pose estimation in addition to training for tasks like instance segmentation and object detection.

B. Analysis-by-Synthesis or Generative Approaches

Analysis-by-synthesis or generative approaches [22]–[26] rely on rendering and verification. They aim to find the best possible explanation for the observed scene by rendering multiple scenes using available 3D models and then finding the best match. Past work on Perception via Search (PERCH) [24]–[26], has demonstrated the capabilities of combining rendering with efficient search for multi-object 3-Dof pose estimation under occlusion and clutter. However PERCH ignores RGB information present in the observed scene as well as in available 3D models. As a result of this, the method fails under some commonly occurring scenarios in homes and retail stores, for example when objects of different brands have the same shape (such as soda cans, cereals etc.). In this work we address this shortcoming and also show that with the help of GPU acceleration, the search runtime can be reduced further than the lazy approach proposed in [25] by an order of magnitude. In addition, we propose a method for RGBD pose estimation in 6-Dof using PERCH 2.0, that combines the strengths of discriminative and generative approaches. On the generative side, it relaxes a few key assumptions made by PERCH, thereby increasing scalability and applicability. On the discriminative side, it allows for 6-Dof prediction directly from the instance segmentation mask and RGBD input, thus eliminating the need of constructing a large dataset consisting of annotated ground truth poses and the need to train additional networks specifically for the prediction of 6-Dof poses of objects.

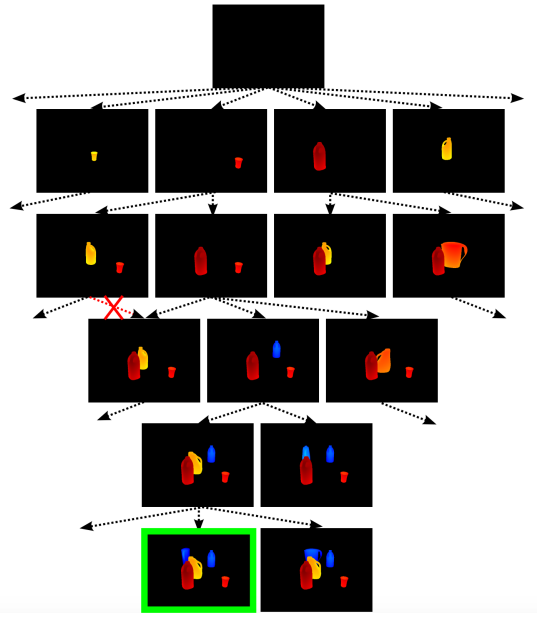

Fig. 2: Portion of the Monotone Scene Generation tree constructed by PERCH [24]. New objects are added as we traverse down the tree. Notice how child states never introduce an object that occludes objects already in the parent state (the red cross shows a counter example). Any state on the Kth level of the tree is a goal state, and the task is to find the one that has the lowest cost path from the root (marked by green bounding box here)

III. Preliminaries

A. Background

Our problem setup and optimization formulation for estimating the 3-DoF pose (x, y, yaw) are similar to those in PERCH [24], which we will re-state here for convenience. We assume a set of K object instances in the input point cloud, given the 3D models of N unique objects. We allow the possibility of cases where multiple copies of a particular object instance are present in the scene. We further assume that the 6-DoF camera pose is given. It may be noted here that our discriminative-generative framework for 6-Dof pose estimation relaxes both assumptions and is described later in Section V. The notations used by us are listed in Table I.

B. Problem Formulation

Given the input point cloud I, PERCH [24] estimates poses of $O_{1:K}$ objects in the scene, by seeking to find a rendered point cloud R_K having K objects, such that every point in I has an associated point in R_K and vice-versa. In other words, PERCH seeks to minimize the following objective :

$$J(O_{1:K}) = \underbrace{\sum_{p \in I} \text{OUTLIER}(p|R_K)}_{J_o(O_{1:K})} + \underbrace{\sum_{p \in R_K} \text{OUTLIER}(p|I)}_{J_r(O_{1:K})} \tag{1}$$

TABLE I: Notations used in PERCH [24] & this work

I	The input point cloud
K	The number of objects in the scene
N	The number of unique objects in the scene ($\leq K$)
O_j	An object state specifying a unique ID and 3-DoF pose
R_K	Point cloud for a rendered scene with K objects $O_{1:K}$
ΔR_j	Point cloud with points belonging exclusively to O_j
$\Delta \tilde{R}_j$	ΔR_j after ICP refinement
$V(O_j)$	The set of points in an admissible (conservative) volume occupied by object O_j, (volume of the inscribed cylinder)
V_j	The union of admissible volumes occupied by objects $O_{1:j}$
H_{rj}	Rotation proposals obtained from sampling for O_j
H_{tj}	Translation proposals obtained from the mask for O_j
$H(O_j)$	6-Dof pose proposals for object O_j
J_o	The observed cost of the scene with respect to given R_j
J_r	The rendered cost of the scene with respect to given R_j

in which OUTLIER$(p|C)$ for a point cloud C and point p is defined as follows:

$$\text{OUTLIER}(p|C) = \begin{cases} 1 & \text{if } \min_{p' \in C} ||p' - p|| > \delta \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where δ is the sensor noise resolution. In order to counter the intractability of this joint global optimization problem owing to a large search comprising of all possible joint poses of all objects, PERCH decomposes the cost function over individual objects added to the rendered scene. The decomposition is subject to the constraint that the newly added object does not occlude those already present. This allows the optimization to be formulated as a tree search problem where a successor state is added to the tree whenever a new object is added to the rendered scene (Figure 2).

It is clear that the expansion of each state in the PERCH search tree has a significant computational cost that scales unfavourably with the number of successors to be generated for the state. Figure 3 illustrates the steps followed during expansion of a state S_1 in the tree. As shown, the successors are generated by first rendering the object O_j to be added to the state in different poses using OpenGL. For each pose, the algorithm then composes the rendered image with an image containing objects already present in the parent state. This step is essential to check if the current object occludes any object already present or to remove pixels corresponding to occlusions caused by other objects in the scene. This is followed by conversion of the rendered depth image to a point cloud and downsampling it, thus obtaining ΔR_j. In order to account for discretization artifacts, local-ICP [33] is used to refine the pose. Since the adjusted state may change its occlusion properties, it is rendered again, composed with the parent image and finally converted to the downsampled adjusted point cloud $\Delta \tilde{R}_j$. k-d tree [34] based nearest neighbour searches are then performed to calculate the observed and rendered cost for each of the successor states. For computing rendered cost J_r, the k-d tree representation of the observed depth input is used and the distance between every point in $\Delta \tilde{R}_j$ and its nearest neighbour in the k-d tree is computed iteratively to classify it as an outlier or inlier according to Equation 2. For observed cost J_o, a similar process is followed, though the k-d tree representation of every $\Delta \tilde{R}_j$ needs to

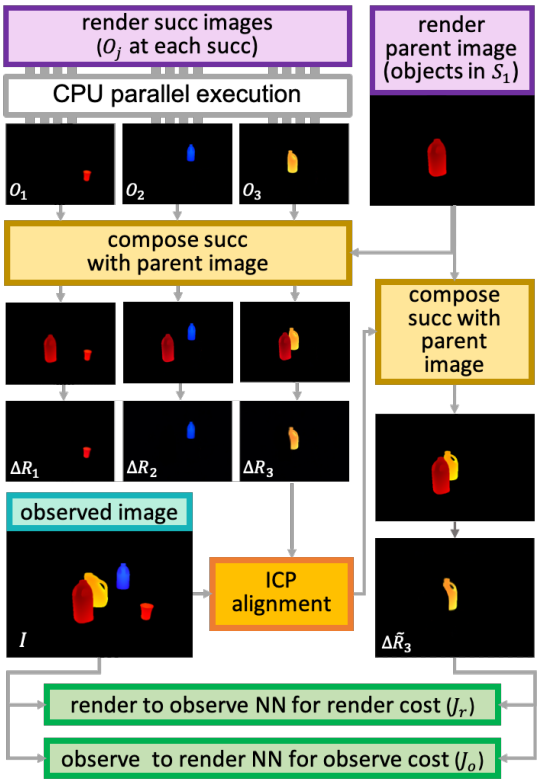

Fig. 3: Expansion of a state S_1 in the PERCH flow on CPU

be constructed. While PERCH uses OpenMPI to exploit the parallelism by executing these sequential steps in parallel threads for each successor state being added to the tree, the restricted number of CPU cores available in regular PCs places a practical limit on the speedup obtained through this approach. Moreover, the approach fails to take advantage of a much wider parallelism in each independent step.

IV. PERCH 2.0

A. GPU Formulation

Parallel Rendering. At a high level, the process of rendering a given number of objects N in state S_1, consisting of P poses of each object can be thought of as having $N \times P$ parallel threads. However if we consider each object and its corresponding 3D mesh model to be made up of T triangles, a parallelism over $N \times P \times T$ threads can be observed. Consider a simple scenario consisting of 4 objects having 10 poses each and 10,000 triangles in each mesh model. The corresponding rendering task exhibits a parallelism of $400,000$ threads. The scale of this parallelism is ideal for exploitation on a GPU and consequently we use that approach in PERCH 2.0. Once the rendered RGB and depth images have been obtained for all objects and poses, they are converted to point clouds on a GPU with every pixel being transformed to its corresponding 3D point using the depth and camera intrinsic parameters in parallel. During this process, we directly produce a downsampled point cloud by downsampling in the image space, reducing a 2 step process to a single one.

Parallel M2M GICP. ICP [33] is an iterative technique to align a given source point cloud to a given target point cloud. PERCH [24] uses a point-to-point non-linear ICP approach from the PCL library. However, this is insufficient to deal with the scalability requirements presented by common pose estimation scenarios. Moreover, a point-to-point ICP approach can lead to low accuracy under high occlusion by converging to the wrong pose. Recent works on GICP [35], [36] have proposed to counter this problem by developing a generalized version of ICP or GICP. GICP combines features of point-to-point and point-to-plane ICP by modelling the surface from which each point is sampled as a Gaussian distribution. We propose a scalable GPU based many to many (M2M) GICP approach that can align several thousand source point clouds to several target point clouds. We use a combination of parallel kNN (described below), batch matrix multiplication from cuBLAS and the linear equation solvers from cuDNN to achieve desired scalibility and speed.

Parallel Cost Computation. The need to create k-d trees for each successor cloud $\Delta\tilde{R}_j$ and then iteratively computing nearest neighbours for every point in every $\Delta\tilde{R}_j$ leads to slow speeds despite the efficiency of the k-d tree data structure. This parallelism over N objects, P poses of each object, consisting of L points in every $\Delta\tilde{R}_j$ can be considered as requiring $N \times P \times L \times I$ parallel threads, where I is the number of points in the input point cloud. We propose to use two approaches to compute the required nearest neighbours which are later compared during evaluation. The first approach (kNN I) [37] fully exploits the underlying parallelism by computing all pairwise distances in parallel. However, it requires the allocation of a large 2D array on the GPU to allow for all threads to simultaneously write to memory locations. This could drive up the peak GPU memory usage and limit the overall number of poses that can be evaluated in parallel. Thus we propose another approach (kNN II) that exploits a reduced parallelism of $N \times P \times L$ threads. In each thread, we loop over the points in I, computing distances to points in $\Delta\tilde{R}_j$ and pushing them into a priority queue. When all threads have finished processing we have the nearest neighbours and corresponding distances between them. Unlike kNN I, the reduced parallelism in this approach limits the memory requirement.

After kNN I or kNN II, another GPU kernel is then used to classify every point as inlier or outlier in parallel, thus obtaining the rendered cost J_r. Finally we use an additional kernel to compute the observed cost J_o, which checks every point in the input scene I and if it lies within the volume occupied by an given object pose $V(O_j)$, simultaneously marking it as inlier or outlier depending on whether it was found as a nearest neighbour for a point in the corresponding $\Delta\tilde{R}_j$ in the previous step.

Parallel Search. Despite speedup from enhancements in the above steps, the runtime remains limited owing to the sequential nature of the Monotone Scene Generation tree. More specifically, the search has to figure out the right non-occluding order in which to place the objects until a solution that satisfies the cost bound has been found. We recount from

Fig. 4: Objects and some sample images from the dataset used for evaluating 3-Dof PERCH 2.0

[24] that this process is primarily a way to model inter-object occlusions. However the work on C-Perch [26] proposed an alternate way to acknowledge inter-object occlusions by marking certain points in the input scene as clutter and use them as extraneous "occluders" while rendering the object of interest in the scene. It was shown that this is incredibly useful when models of all objects in the scene are not available and thus the Monotone Scene Generation tree can't be used to account for all inter-object occlusions. We build on this strategy in PERCH 2.0, by treating the search for each object as an independent search for that object in a cluttered scene where the model for other objects is unknown. This change effectively reduces a sequential search to a parallel one that can be performed efficiently with our GPU based pipeline. From [26], we also note the changes to the terms J_o and J_r in Equation 1 :

$$\begin{aligned} J_o\left(O_{1:K}\right) &= \textstyle\sum_{p \in I \cap V_K} \text{OUTLIER}\left(p \,|(R_K \setminus C_K)\right) \\ J_r\left(O_{1:K}\right) &= \textstyle\sum_{p \in R_K \setminus C_K} \text{OUTLIER}(p|I) \end{aligned} \tag{3}$$

Here C_K represents the extraneous "occluders" that occlude the scene created by rendering the object poses $O_{1:K}$ and $R_k \setminus C_K$ is the corresponding scene point cloud with C_K removed. Following a strategy similar to [26] for creating $R_k \setminus C_K$ by using the input depth image, we render and compute costs for all successors and find the one corresponding to the minimum cost for each object in parallel on the GPU.

B. RGBD Cost Formulation

The formulation of explanation cost in PERCH is based on the implicit assumption that depth data alone can be used to capture how well a rendered point cloud matches the observed point cloud. More formally, the classification of a point p in a point cloud C as an outlier in Equation 2 is entirely based on the Euclidean distance between them in 3D space. However this definition fails in scenarios similar to those depicted in Figure 4. In such scenarios, where objects of similar shape are present, PERCH is unable to estimate the (x, y, yaw) correctly because rendering any object at a

given (x, y, yaw) results in the same change in cost, owing to an outlier definition based purely on Euclidean distance.

Intuitively, the explanation cost in such cases must utilise the difference in point-wise RGB information present in ICP adjusted rendered clouds $\Delta\tilde{R}_j$ and in the observed point cloud. It must also accommodate changes in perceived color due to lighting. Keeping these requirements in mind, we introduce the CIEDE2000 color difference formula [38] in the CIELAB color space to perform the comparison between a point in the observed cloud I and the rendered point cloud ΔR_j or vice-versa. In this space, each color is represented by 3 values - $L*$, $a*$ and $b*$ and uniform changes in these components are designed to replicate uniform changes in color as perceived by the human eye. Formally, for a point p, the OUTLIER$(p|C)$ definition in Equation 2 can be re-written as :

$$\text{OUTLIER\_RGBD}(p|C) = \begin{cases} 1 & \text{if } \min_{p' \in C} ||p' - p|| > \delta \\ 1 & \text{if } ||p''_c - p_c||_c > \tau_c \\ & \text{s.t. } \min_{p'' \in C} ||p'' - p|| \leq \delta \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where :

- p_c and p''_c denote the color in CIELAB of points p and p'' respectively
- $||p''_c - p_c||_c$ denotes the CIEDE2000 [38] color-difference between the two points
- τ_c denotes the maximum allowed color difference for two colors to be considered same

With this definition of OUTLIER_RGBD$(p|C)$, we penalize points for being distant in color space even though they might satisfy the Euclidean constraint in 3D space.

V. PERCH 2.0 for 6-Dof Pose Estimation

A. Formulation

Due to the success of CNNs in 2D instance segmentation and the search speedup achieved by GPU in PERCH 2.0, we have the opportunity to extend perception via search to 6-Dof by using the CNN output to generate 6-Dof pose proposals which are then evaluted by PERCH 2.0.

B. Pose Proposal Generation

Rotation Proposals. Following the work in SSD-6D [20], we represent all possible rotations as a set of viewpoints v and in-plane rotation angles θ. We sample M equidistant viewpoints from unit sphere and N in-plane rotation angles from $[0, 2\pi]$ and combine each with the other to generate $M \times N$ possible rotation proposals for object O_j:

$$H_{rj} = \langle v_i, \theta_k \rangle \text{ where } 1 \leq i \leq M \text{ and } 1 \leq k \leq N \tag{5}$$

Translation Proposals. We generate a set of translation proposals for object O_j as follows:

$$H_{tj} = \langle x_c, y_c, z_i \rangle \text{ where } z_{min} \leq z_i \leq z_{max} \tag{6}$$

In the above equation, $\langle x_c, y_c \rangle$ is obtained by back projecting the center of object's 2D bounding box into 3D space using the camera's projection matrix. In our framework, we propose to detect the "full" 2D bounding box [39] as opposed to the typical "visible" bounding box. A "full" bounding box assists in pose estimation of occluded objects by giving us a more accurate location of the bounding box center. z_i ranges from z_{min} , the closest point to the camera corresponding to the given object in the observed depth image to, z_{max} , the farthest point from the camera corresponding to the given object in the observed depth image. These are obtained by combining the segmentation mask for the object with the input depth image.

6-Dof Pose Proposals. H_{rj} and H_{tj} are combined with each other to create 6D pose hypotheses for every object $H(O_j) = H_{rj} \cdot H_{tj}$.

C. 6-Dof Pose Estimation Pipeline

A pictorial representation of the entire pipeline can be seen in Fig 5. The input RGB image is passed through a MaskRCNN [40] instance segmentation network, obtaining object labels, segmentation masks and "full" bounding boxes. Then we generate 6-Dof pose proposals for the detected objects through parallel rendering of each pose proposal on GPU using the method describe in V-B and IV-A. While marking points as extraneous clutter, we use the class labels of the pixel to make sure that the occluders belong to a different object than the one being rendered. We then generate point clouds which are refined using the proposed parallel M2M GICP approach.

Finally, we render and generate point clouds for the adjusted poses and compute the cost of each pose proposal in parallel. For calculating of J_o in Equation 3, instead of explicitly computing $V(O_j)$, the pixel-wise segmentation labels are used directly to determine the set of observed points belonging to a given object.

VI. Evaluation

A. PERCH 2.0 for 3-Dof Pose Estimation

Dataset. Early experimentation revealed that PERCH can exploit minute differences in shape and estimate poses accurately. Thus, for evaluating PERCH 2.0 against PERCH, we focus on images of common objects that have same shape but different appearance. Such objects are commonly found in grocery stores but to our knowledge, no dataset exists in the literature that consists of RGBD images of such objects. Moreover for PERCH, we require variation only in 3-Dof pose (x, y, yaw) for every object while common annotated pose estimation databases consist of pose variation in 6-Dof. Subsequently, we constructed a synthetic photo-realistic dataset of 75 scenes with corresponding RGBD images using the recently released NVidia NDDS [41] plugin for Unreal Engine 4 (objects shown in Figure 4). Within the plugin, we randomly vary 3D pose (x, y, yaw) of every object on a tabletop while keeping (z, roll and pitch) constant. The plugin allows generation of images with realistic lighting conditions and inter-object occlusion.

Baselines. We compare results of PERCH 2.0 with *PERCH* and *DOPE [19] + ICP*. *DOPE* is a leading RGB

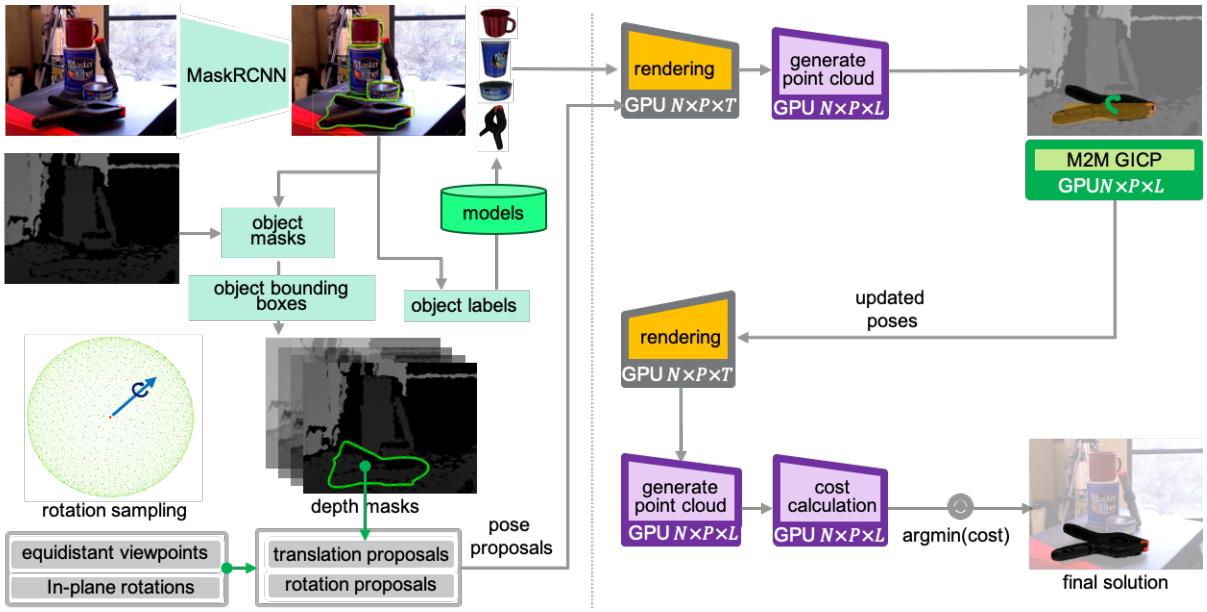

Fig. 5: 6-Dof Pose Estimation Pipeline (N : Number of objects, P : Number of poses per object, T : Number of triangles in an object model, L : Number of points in given pose point cloud)

based 6-Dof pose estimation method directly compatible with NDDS generated data which we combine with ICP refinement on the depth input for our experiments. For training *DOPE*, we construct a training dataset of total 12K images containing each of the 6 objects using NDDS [41]. The network was trained for 60 epochs (pretrained on ImageNet) on each object individually, taking approximately 12 hours for each on 2 NVidia P100 GPUs. The Brute Force ICP (*BF-ICP*) baseline proposed in [24] is also used for comparison. Further, in order to understand the effectiveness of occlusion handling and impact of having full parallelization, we use following additional variants of *PERCH 2.0* : 1) *PERCH 2.0-A* which doesn't use the input depth data to mark occluded points in the rendered scenes, 2) *PERCH 2.0-B* which doesn't use full parallelization like *PERCH 2.0* but instead uses the Monotone Scene Generation tree formulation. All variants use kNN-I and the same point-to-point ICP used by *PERCH*. For inference and other evaluation experiments, a machine with 8 CPU cores and an NVidia GTX 1070 8GB GPU is used. For *PERCH* and *PERCH 2.0*, we use a translation discretization of 0.08 m and a yaw discretization of 22.5 degrees. The sensor resolution δ is set to 0.0075 m. The CIEDE2000 color difference threshold τ_c is set to 12.5.

Metrics. We use the ADD-S [13], [28] metric for evaluation which computes the average distance between the closest points in the object's 3D model, transformed with ground truth pose and the same model transformed with the predicted pose. We vary the ADD-S distance threshold up to 0.1 m and obtain the area under the accuracy-threshold curve (AUC) for all methods as shown in Table II. We also compute ADD-S$<$1cm, which denotes the percentage of poses with less than 1cm ADD-S error.

Accuracy. *PERCH 2.0-C* achieves the best performance among all variants with 100% of poses below ADD-S 1cm error. It can also be noted that *PERCH 2.0* variants and *DOPE + ICP* outperform *PERCH* and *BF-ICP*. This shows that *PERCH 2.0* and *DOPE* are able to utilise the RGB information present in the object model and observed scene and closer inspection reveals that these methods don't get confused between similar looking objects even in occlusion (like sprite_can and pepsi_can).

Robustness. The robustness of the RGBD cost function used by *PERCH 2.0-C* is highlighted by its ability to differentiate between objects of different sizes (bottle vs can), objects with minute color differences (pepsi can vs sprite can) and objects with a non-uniform color distribution (sprite can, 7up can). *PERCH 2.0-C* also handles occlusions more effectively as compared to *DOPE + ICP* and *PERCH 2.0-A*, which is exhibited in its better performance as compared to both.

Runtime. From Table II it is clear that we are able to achieve an order of magnitude improvement in runtime with *PERCH 2.0-C* over *PERCH* ($\sim$100X). A comparison between *PERCH 2.0-C* and *PERCH 2.0-B* also reveals that *PERCH 2.0-C* is able to achieve the same accuracy with full parallelization that *PERCH 2.0-B* is able to obtain using the Monotone Scene Generation tree [24]. However *PERCH 2.0-C* is 10 times faster than the latter. Moreover, *PERCH 2.0-C* has a runtime close to the *DOPE + ICP* pipeline which suggests that it can achieve speeds comparable to popular learning based approaches followed by depth-based refinement without requiring any training for estimating 3-Dof poses and object categories.

B. PERCH 2.0 for 6-DoF Pose Estimation

Baselines. In order to evaluate the performance of PERCH 2.0 for 6-Dof pose estimation, we compare our results with *DenseFusion* [42] and *PoseCNN + ICP* [13] on objects from the YCB-Video Dataset [13]. The results are computed for the 2,949 keyframes used for testing in prior works.

TABLE II: Area under accuracy-threshold (ADD-S) curves for 3-Dof pose estimation

Objects	BF-ICP [24]		PERCH [24]		DOPE [19] + ICP		PERCH 2.0-A (W/O Occluder Marking)		PERCH 2.0-B (W/O Full Parallelization)		PERCH 2.0-C	
	AUC	<1cm	AUC	<1cm	AUC	<1cm	AUC	<1cm	AUC	<1cm	AUC	<1cm
coke_bottle	46.61	0.00	55.43	58.00	90.00	94.00	96.59	100.0	96.6	100.00	**96.59**	**100.00**
sprite_bottle	46.16	0.00	55.37	58.00	87.99	84.44	97.06	100.0	96.65	100.00	**97.09**	**100.00**
sprite_can	17.62	0.00	43.04	30.00	90.71	80.00	57.41	60.00	95.42	100.00	**95.61**	**100.00**
pepsi_can	38.10	0.00	48.63	48.57	94.82	96.00	95.66	100.0	95.63	100.00	**95.69**	**100.00**
coke_can	46.61	0.00	40.58	40.00	89.18	89.18	93.39	97.30	95.61	100.00	**95.95**	**100.00**
7up_can	28.27	0.00	32.46	25.00	75.21	68.00	79.33	68.00	95.03	100.00	**95.26**	**100.00**
All Objects	37.51	0.00	47.16	43.26	88.16	85.27	80.49	87.55	95.26	100.00	**95.72**	**100.00**
Mean Runtime (s)	220.7		137.2		**1.0**		1.64		11.9		1.31	

TABLE III: Area under accuracy-threshold curves for 6-Dof pose estimation on objects from the YCB Video Dataset [13]

Objects	PoseCNN + ICP [13]		DenseFusion (Per-Pixel) [42]		DenseFusion (Iterative) [42]		PERCH 2.0-A (PoseCNN Mask)		PERCH 2.0-B (MaskRCNN Mask)	
	AUC	<2cm	AUC	<2cm	AUC	<2cm	AUC	<2cm	AUC	<2cm
002_master_chef_can	95.80	100.00	95.20	100.00	**96.40**	100.00	96.06	100.00	96.25	**100.00**
003_cracker_box	92.70	91.60	92.50	99.30	**95.50**	99.50	93.54	97.81	94.69	**99.65**
004_sugar_box	**98.20**	100.00	95.10	100.00	97.50	**100.00**	95.86	99.66	96.11	99.58
005_tomato_soup_can	94.50	96.90	93.70	96.90	94.60	96.90	97.26	99.77	**97.30**	**100.00**
006_mustard_bottle	**98.60**	100.00	95.90	100.00	97.20	100.00	97.51	100.00	97.42	**100.00**
007_tuna_fish_can	**97.10**	100.00	94.90	100.00	96.60	100.00	95.50	99.91	95.97	**100.00**
008_pudding_box	**97.90**	100.00	94.70	100.00	96.50	**100.00**	93.04	94.03	93.55	99.53
009_gelatin_box	**98.80**	100.00	95.80	100.00	98.10	100.00	96.77	100.00	96.56	**100.00**
010_potted_meat_can	92.70	93.60	90.10	93.10	91.30	93.10	95.13	97.82	**95.45**	**99.72**
011_banana	**97.10**	99.70	91.50	93.90	96.60	**100.00**	96.53	99.74	96.88	99.74
019_pitcher_base	**97.80**	100.00	94.60	100.00	97.10	100.00	92.37	100.00	92.11	**100.00**
021_bleach_cleanser	**96.90**	99.40	94.30	99.80	95.80	100.00	93.39	96.99	95.25	**100.00**
024 bowl	81.00	54.90	86.60	69.50	88.20	98.80	93.42	97.04	**97.22**	**100.00**
025_mug	95.00	99.80	95.50	100.00	**97.10**	100.00	96.96	100.00	96.96	**100.00**
035_power_drill	**98.20**	99.60	92.40	97.10	96.00	98.70	96.10	**99.91**	95.72	99.72
036_wood_block	87.60	80.20	85.50	93.40	89.70	**94.60**	90.31	90.08	**91.58**	93.61
037_scissors	91.70	95.60	96.40	100.00	95.20	100.00	95.11	100.00	**96.49**	**100.00**
040_large_marker	97.20	99.70	94.70	99.20	97.50	100.00	97.56	99.85	**97.78**	**100.00**
051_large_clamp	75.20	74.90	71.60	78.50	72.90	79.20	72.25	77.06	**92.41**	**97.99**
052_extra_large_clamp	64.40	48.80	69.00	69.50	69.80	76.30	86.12	82.58	**88.54**	**90.24**
061_foam_brick	**97.20**	100.00	92.40	100.00	92.50	100.00	95.89	100.00	95.72	**100.00**
All Objects	93.00	93.20	91.20	95.30	93.10	96.80	94.56	98.00	**95.48**	**99.29**

We use two variants of PERCH 2.0 for evaluation : 1) *PERCH 2.0-A* uses the PoseCNN segmentation masks published published online [1] and also used by DenseFusion. The required bounding box is computed from the mask boundaries. We note that this is the "visible" bounding box. 2) *PERCH 2.0-B* uses a MaskRCNN [40], [43] model trained by us on the YCB-Video Dataset. Since the YCB-Video dataset doesn't contain "full" bounding boxes annotations, we use the ground truth 6-Dof pose and project it onto the image to obtain the "full" bounding box annotations used to train the model. The training is performed on 4 NVidia V100 GPUs. We note that "full" bounding box annotations could also be obtained through crowdsourced human annotation as done for the CrowdHuman [39] dataset. Both variants use kNN II and the proposed M2M GICP framework.

Accuracy. The results of our evaluation are shown in Table III for ADD-S<2cm and ADD-S AUC (<0.1m). We can observe that even with the use of PoseCNN mask and "visible" bounding boxes, *PERCH 2.0-A* outperforms *DenseFusion* and *PoseCNN + ICP* baselines. We observe that *PERCH 2.0-B* which uses "full" bounding boxes further improves on the accuracy and performs well across objects of varying shape, size, texture, symmetry & visibility, estimating 99.29% of the poses below 2cm ADD-S error and hence within the tolerance limit of most robot grippers.

[1] https://rse-lab.cs.washington.edu/projects/posecnn/

TABLE IV: Evaluation of runtime on YCB Video Dataset

Method	Average Runtime (s)
PoseCNN + ICP	10.00
DenseFusion (Iterative)	**0.06**
PERCH 2.0-A (kNN I + CPU GICP)	75.43
PERCH 2.0-B (kNN II + M2M GICP)	7.60

Runtime. We use two variants of PERCH 2.0 for runtime evaluation : 1) *PERCH 2.0-A* which uses kNN I and the publicly available CPU parallelized version of GICP [36]. 2) *PERCH 2.0-B* which uses kNN II and our proposed parallel M2M GICP approach. The experiments are performed on a machine with 32 CPU cores and a NVidia P100 16GB GPU. From Table IV, we can observe that *PERCH 2.0-B* takes only 7.6s on average to estimate poses for all objects in the scene.

It achieves a $\sim$10X runtime improvement over *PERCH 2.0-A*, highlighting the importance of parallel M2M GICP and kNN II when the number of poses to be evaluated is high. For both variants, an average of 2400 poses are evaluated per scene for all objects combined. We note that the runtime of *PERCH 2.0-B* is even lower than *PoseCNN + ICP* even though we don't use a CNN for estimating the final 6-Dof pose.

VII. Conclusion

In this work we introduced PERCH 2.0, a novel generative GPU-based perception via search technique that achieves an order of magnitude improvement in runtime over its predecessor PERCH. PERCH 2.0 seamlessly incorporates RGB input along with depth in its cost function to enhance its accuracy. We also presented a combined discriminative-generative framework for 6-Dof pose estimation that outperforms state-of-the-art purely discriminative approaches but doesn't require training with ground truth pose annotation.

VIII. Acknowledgment

This work was supported by ARL grant W911NF-18-2-0218 as part of the A2I2 program.

References

[1] A. Aldoma, M. Vincze, N. Blodow, D. Gossow, S. Gedikli, R. B. Rusu, and G. Bradski, "CAD-model recognition and 6DOF pose estimation using 3D cues," in *ICCV Workshops*, 2011, pp. 585–592.

[2] A. Aldoma, F. Tombari, R. B. Rusu, and M. Vincze, "OUR-CVFH–oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6DOF pose estimation," in *Joint DAGM and OAGM Symposium*, 2012, pp. 113–122.

[3] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce, "3D object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints," *IJCV*, pp. 231–259, 2006.

[4] M. R. Stevens and J. R. Beveridge, *Integrating graphics and vision for object recognition*. Springer Science & Business Media, 2000.

[5] P. Wohlhart and V. Lepetit, "Learning descriptors for object recognition and 3D pose estimation," in *CVPR*, 2015, pp. 3109–3118.

[6] S. Tulsiani and J. Malik, "Viewpoints and keypoints," in *CVPR*, 2015, pp. 1510–1519.

[7] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, "6-DOF object pose from semantic keypoints," in *ICRA*, 2017, pp. 2011–2018.

[8] J. Liu and S. He, "6D Object Pose Estimation without PnP," *CoRR*, 2019.

[9] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3D bounding box estimation using deep learning and geometry," in *CVPR*, 2017, pp. 7074–7082.

[10] C. Mitash, K. E. Bekris, and A. Boularias, "A self-supervised learning system for object detection using physics simulation and multi-view pose estimation," in *IROS*, 2017, pp. 545–551.

[11] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, "Implicit 3D orientation learning for 6D object detection from rgb images," in *ECCV*, 2018, pp. 699–715.

[12] S. Suwajanakorn, N. Snavely, J. J. Tompson, and M. Norouzi, "Discovery of latent 3D keypoints via end-to-end geometric reasoning," in *NIPS*, 2018, pp. 2059–2070.

[13] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," in *RSS*, 2018.

[14] B. Tekin, S. N. Sinha, and P. Fua, "Real-time seamless single shot 6D object pose prediction," in *CVPR*, 2018, pp. 292–301.

[15] M. Rad and V. Lepetit, "BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth," *CoRR*, 2017.

[16] E. Corona, K. Kundu, and S. Fidler, "Pose Estimation for Objects with Rotational Symmetry," in *IROS*, 2018, pp. 7215–7222.

[17] C. Mitash, A. Boularias, and K. E. Bekris, "Improving 6D pose estimation of objects in clutter via physics-aware Monte Carlo tree search," in *ICRA*, 2018, pp. 1–8.

[18] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, "Densefusion: 6d object pose estimation by iterative dense fusion," in *CVPR*, 2019, pp. 3343–3352.

[19] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects," in *CoRL*, 2018.

[20] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again," in *ICCV*, 2017, pp. 1521–1529.

[21] P. Marion, P. R. Florence, L. Manuelli, and R. Tedrake, "Label Fusion: A Pipeline for Generating Ground Truth Labels for Real RGBD Data of Cluttered Scenes," in *ICRA*, 2018, pp. 3235–3242.

[22] M. Stevens and J. Beveridge, "Localized Scene Interpretation from 3D Models, Range, and Optical Data," *Computer Vision and Image Understanding*, pp. 111–129, 2000.

[23] A. Aldoma, F. Tombari, L. Di Stefano, and M. Vincze, "A global hypotheses verification method for 3D object recognition," in *ECCV*, 2012, pp. 511–524.

[24] V. Narayanan and M. Likhachev, "PERCH: Perception via search for multi-object recognition and localization," in *ICRA*, 2016, pp. 5052–5059.

[25] ——, "Discriminatively-guided Deliberative Perception for Pose Estimation of Multiple 3D Object Instances." in *RSS*, 2016.

[26] ——, "Deliberative object pose estimation in clutter," in *ICRA*, 2017, pp. 3125–3130.

[27] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, pp. 40–53, 2008.

[28] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes," in *ACCV*, 2012, pp. 548–562.

[29] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit, "Gradient response maps for real-time detection of textureless objects," *PAMI*, pp. 876–888, 2011.

[30] Z. Cao, Y. Sheikh, and N. K. Banerjee, "Real-time scalable 6DOF pose estimation for textureless objects," in *ICRA*, 2016, pp. 2441–2448.

[31] X. Deng, A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox, "Poserbpf: A rao-blackwellized particle filter for 6d object pose tracking," *CoRR*, 2019.

[32] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, "Implicit 3d orientation learning for 6d object detection from rgb images," in *ECCV*, 2018.

[33] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and vision computing*, pp. 145–155, 1992.

[34] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *ACM*, pp. 509–517, 1975.

[35] A. V. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP," in *RSS*, 2009.

[36] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, "Voxelized gicp for fast and accurate 3d point cloud registration," EasyChair Preprint no. 2703, 2020.

[37] V. Garcia, Debreuve, F. Nielsen, and M. Barlaud, "K-nearest neighbor search: Fast gpu-based implementations and application to high-dimensional feature matching," in *ICIP*, 2010.

[38] G. Sharma, W. Wu, and E. N. Dalal, "The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations," *Color Research & Application*, pp. 21–30, 2005.

[39] S. Shao, Z. Zhao, B. Li, T. Xiao, G. Yu, X. Zhang, and J. Sun, "Crowdhuman: A benchmark for detecting human in a crowd," *arXiv preprint arXiv:1805.00123*, 2018.

[40] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," in *ICCV*, 2017, pp. 2980–2988.

[41] T. To, J. Tremblay, D. McKay, Y. Yamaguchi, K. Leung, A. Balanon, J. Cheng, W. Hodge, and S. Birchfield, "NDDS: NVIDIA deep learning dataset synthesizer," 2018.

[42] C. Wang, D. Xu, Y. Zhu, R. Martin-Martin, C. Lu, L. Fei-Fei, and S. Savarese, "DenseFusion: 6D object pose estimation by iterative dense fusion," in *CVPR*, 2019, pp. 3338–3347.

[43] F. Massa and R. Girshick, "maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch," 2018.