Special Issue: Robotics Collaborative Technology Alliance (RCTA) Program

Regular Article

Fast and high-quality, GPU-based, deliberative, object-pose estimation

Aditya Agarwal[1], Yash Oza[1], Maxim Likhachev[1] and Chad C. Kessens[2]
[1]The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
[2]US Army Research Laboratory, Aberdeen Proving Ground, Maryland, USA

Abstract: Pose estimation of recognized 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 and object templates in order to find correspondence between models and observed data. However, these methods require extensive annotation of ground-truth poses. An alternative is to use algorithms, such as PERCH (PErception Via SeaRCH) that seek an optimal explanation of the observed scene in a space of possible rendered versions. While PERCH offers strong guarantees on accuracy, the initial formulation suffers from poor scalability owing to its high runtime. In this work, we present PERCH 2.0, a deliberative approach that takes advantage of GPU acceleration and RGB data by formulating pose estimation as a single-shot, fully parallel approach. We show that PERCH 2.0 achieves a two orders of magnitude speedup (∼100X) over the hierarchical PERCH by evaluating thousands of poses in parallel. In addition, we propose a combined deliberative and discriminative framework for 6-DoF pose estimation that doesn't require any ground-truth pose-annotation. Our work shows that PERCH 2.0 achieves, on the YCB-Video Dataset, a higher accuracy than DenseFusion, a state-of-the-art, end-to-end, learning-based approach. We also demonstrate that our work leads directly to an extension of deliberative pose estimation methods like PERCH to new domains, such as conveyor picking, which was previously infeasible due to high runtime. Our code is available at https://sbpl-cruz.github.io/perception/

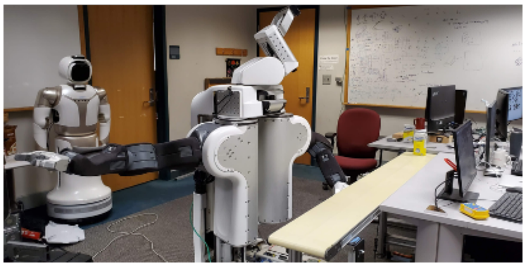
Keywords: perception, manipulators, computer vision, pose estimation

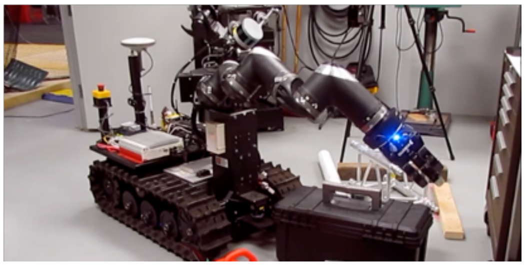
1. Introduction

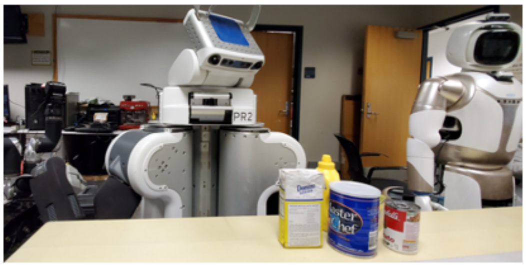
For robotic manipulators to operate successfully outside controlled environments, they need to be able to interact with objects in a safe and reliable manner. Such interaction requires correct identification of object categories as well as their location and orientation in the 3D world. Manipulators are often deployed in settings, such as an automated warehouse, where the objects of interest could either be at rest in shelves or in bins or moving, as on a conveyor (Figure 1a). In either scenario, robust

Received: 20 September 2020; revised: 28 February 2021; accepted: 24 April 2021; published: 22 November 2021.

Correspondence: Aditya Agarwal, The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA, Email: adityaa2@alumni.cmu.edu


(a) Conveyor Picking


(b) Container Opening



(c) Table top robotic manipulation

Figure 1. Examples of robotic manipulation scenarios.

recognition and pose estimation of the objects in the 3D world could greatly enhance the capabilities of the deployed manipulator. The knowledge of the object pose allows the underlying planning system to retain the flexibility of choosing an optimal grasp for picking up the object while avoiding collisions with other objects or the environment.

Pick and place tasks performed by full body mobile manipulators appear in household environments as well. The objects could be located on a table top (Figure 1c), on shelves, or within household enclosures such as cabinets, dishwashers, refrigerators etc. More recently, research has also focused on object articulation with the help of mobile robotic manipulators, especially in situations where such handling would be dangerous for human beings (e.g. in an active war zone). One such task that involved opening of a container and removing an object within for inspection, was the focus of a project by the US Army Research Laboratory's (ARL's) Robotics Collaborative Technology Alliance (RCTA) (Figure 1b) using the RoMan platform (Kessens et al., 2020).

A key difference in the domains described above is the degree of structure they present. Objects present in automated warehouses and grocery stores on shelves are often arranged in a pre-decided orientation (such as upright or lying down). In addition, an object's location in a store, for example, can give a hint of object's identity or category. Similarly, in the case of container opening, the container to be located can vary only in 3 degrees of freedom (DoF) that is (x, y, yaw). However, in other domains, such as household environments, the objects could be found in random orientations and locations, varying in all 6-DoF (x, y, z, roll, pitch, yaw). The identity of the objects would also be independent of their location.

Despite differing domain-dependent requirements for interaction with objects, several challenges in the underlying pose estimation task are common:

- Objects are often occluded by other objects or by unknown obstacles in the environment.
- The manipulator itself could be fixed or dynamic (e.g. mounted to a mobile base), the latter requiring the algorithm to be robust to different background scenes and camera viewpoints.
- For picking moving objects from a conveyor, a fast pose estimation approach that provides high accuracy when the object is at a distance is needed to plan and execute a motion to pickup the

object. Similarly, tasks like container opening, performed in military settings, are time critical and require fast pose estimation.

In this paper, we build on prior work such as PERCH (Narayanan & Likhachev, 2016a, 2016b, 2017) and combine advancements in computer vision, search, and optimization techniques with advancements in computing hardware to devise an approach that scales with different domain requirements. The primary focus areas of our approach (referred to as PERCH 2.0 (Agarwal et al., 2020)) include ease of use, ready interpret-ability and the rapid adaptability for unseen objects encountered in the field, all while maintaining the the accuracy of the estimated poses especially under occlusion, since it has a direct positive impact on robotic manipulation success. Our approach also seeks to exploit structure, wherever present in the environment, for improved speed and reduced resource requirements. We also note that PERCH 2.0 has been used by recent works (Saxena et al., 2021) to perform real world manipulation. The key contributions of our work can be listed as follows:

- PERCH 2.0: A fully parallel and scalable GPU-based deliberative pose estimation methodology that achieves significant speedup over previous deliberative methods such as PERCH
- A combined discriminative and deliberative framework for 6-Dof pose estimation that eliminates the need for ground-truth pose-annotation in the training data and overcomes other shortcomings of deliberative methods by using a discriminative CNN to prune large parts of the search space
- A GPU-based, generalized ICP (GICP) approach to refine large number 6-Dof pose proposals in parallel
- Incorporation of RGB sensor data into the objective function used by PERCH for 3-Dof pose estimation, allowing the algorithm to handle scenarios where depth data alone is not sufficient to estimate the object poses
- Demonstration of the application of PERCH 2.0 for pose estimation in new domains such conveyor picking and container opening, as a result of improved runtime

2. Related work

In terms of methodology, approaches that address pose estimation of objects in 3D space can be broadly divided into 2 categories: discriminative methods and deliberative methods.

2.1. Discriminative methods

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 (Aldoma, Tombari, Rusu, et al., 2012; Aldoma et al., 2011; Johnson & Hebert, 1999; Lowe, 1999; Rothganger et al., 2006; Rusu et al., 2009; Rusu et al., 2010; Tombari et al., 2010). 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 (Cao et al., 2016; Hinterstoisser et al., 2012; Hinterstoisser et al., 2013). Feature-based methods 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 individual object poses in isolation may not lead to a globally feasible and optimal solution that fully explains the observed scene (Stevens & Beveridge, 2000a).

Recent advancements in deep learning have led to the extension of 2D object detectors for the task of 6-Dof pose estimation (Corona et al., 2018; Kehl et al., 2017; Liu & He, 2019; Mitash et al., 2017; Mousavian et al., 2017; Pavlakos et al., 2017; Rad & Lepetit, 2017; Sundermeyer et al., 2018; Suwajanakorn et al., 2018; Tekin et al., 2018; Tremblay et al., 2018; Tulsiani & Malik, 2015; Wang et al., 2019; Wohlhart & Lepetit, 2015; Xiang et al., 2018). These approaches can further be divided into: methods that directly regress to 6-Dof pose coordinates and rotations (Billings & Johnson-Roberson, 2019; Wang et al., 2019; Xiang et al., 2018), methods that localize object keypoints (Pavlakos et al., 2017; Rad & Lepetit, 2017; Suwajanakorn et al., 2018; Tekin et al., 2018; Tremblay et al., 2018; Tulsiani & Malik, 2015) in the image space or methods that discretize the 6D space and then score each discretized pose using a classifier (Corona et al., 2018; Kehl et al., 2017; Mitash et al., 2018). Regressing directly to poses ties the pose estimation to camera intrinsics, thus introducing errors if the camera is changed. Localization to image keypoints often results in ambiguities for objects with symmetries or requires explicit handling of symmetries. Scoring discretized poses is independent of camera parameters and object symmetries but requires post prediction refinement to arrive at the final pose. Moreover, several methods in each of these categories require extensive annotation of ground-truth 6-Dof poses in the training data. While large scale annotation for bounding boxes and instance segmentation masks can be automated or crowd-sourced through online tools such as Amazon's Mechanical Turk (Buhrmester et al., 2011), preparation of 6-Dof pose labels requires specialized offline tools such as Labelfusion (Marion et al., 2018). Dataset sizes scale poorly with number of objects since networks need to be trained on multiple objects from as many viewpoints as possible per object with varying degrees of inter-object occlusions.

Other methods like Mitash et al. (2017) and Tremblay et al. (2018) use synthetic data to counter the annotation requirement while authors in Deng et al. (2021) and Sundermeyer et al. (2018) train an auto-encoder and use the encoder embedding space to compute similarity scores between observed image and rendered images constructed by uniformly sampling the rotation space. However, these methods still require training of a pose estimation neural network in addition to training for standard computer vision tasks like instance segmentation and object detection. In Deng et al. (2020), the authors explore a self-supervised approach to generate annotated pose estimation data that could be used by learning based techniques for continuous improvement. However the method requires pose initialization from an existing technique as well as a robot to generate poses from multiple viewpoints.

2.2. Deliberative methods

Analysis-by-synthesis or deliberative approaches (Aldoma, Tombari, Di Stefano, et al., 2012; Mitash et al., 2018; Narayanan & Likhachev, 2016a, 2016b, 2017; Stevens & Beveridge, 2000b; Wong et al., 2017) rely on rendering and verification. Past work on Perception via Search (PERCH) (Narayanan & Likhachev, 2016a, 2016b, 2017), demonstrated the effectiveness of combining rendering with search for multi-object 3-Dof pose (x, y, yaw) estimation under occlusion and clutter. However, the initial formulation's high runtime and poor scalability, restricts its application to only 3-Dof pose estimation of static objects. In addition, the formulation ignores RGB information present in the observed scene as well as in available 3D models. As a result of this, the method fails under 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.).

D2P (Narayanan & Likhachev, 2016a), an extension to PERCH used the output of a 2D R-CNN object detector as a heuristic to speed up the tree search. Similarly, the work in Mitash et al. (2018) uses the output of a 2D object detector to construct a 6D pose hypothesis which is then evaluated through a search. However, none of these approaches exploit the underlying parallelism in hypothesis evaluation and are thus limited by high run times. The authors in Wong et al. (2017) improve run time using a similar approach and tracking poses over time instead of re-evaluating the hypothesis. Research works using traditional pose estimation techniques such as Choi and Christensen (2016) employ GPU parallelization to speed up their pose estimation algorithms. However in recent years the use of GPUs has been largely restricted to end-to-end learning algorithms. In our work, we show that a deliberative-discriminative 6-Dof pose estimation framework, where the deliberative component takes advantage of GPU parallelization can achieve a low run time as well as require minimal pose-annotation. Such integration has been explored in early research works such as Mörwald et al. (2010), and has only recently started to receive focus from the research community in the deep learning era (Labbé et al., 2020).

2.3. Pose refinement methods

ICP (Chen & Medioni, 1992) is a popular optimization approach that is used in pose estimation as well as tasks such as, scan matching and tracking camera pose of a robot in motion. Several enhanced versions of the original ICP algorithm have been proposed over the years, such as, the probabilistic Generalized ICP (GICP) (Segal et al., 2009). GICP models surface of different types through Gaussian distributions. Despite increase in accuracy and speed of recent ICP variants (Koide et al., 2021), ICP alone is insufficient for object pose estimation in clutter since it requires initialization with a pose estimate close to the ground-truth. However, this makes ICP a perfect candidate for refinement of poses, predicted through other methods (Wong et al., 2017; Xiang et al., 2018). In this process, special care must be taken to avoid local minima.

ICP is an important component of PERCH (Narayanan & Likhachev, 2016b) that helps resolve pose discretization artifacts that arise from rendering of the pose hypotheses. However as we will show in our work, the ICP approach utilized in PERCH and other methods, doesn't scale well with number of objects and number of poses being considered. In this work, we extend ideas proposed for GPU-based ICP and nearest neighbour search parallelization in Garcia et al. (2010), Koide et al. (2021), and Qiu et al. (2009), so that several thousand poses can be refined using ICP in parallel.

3. Background

This section states the problem setup and optimization formulation for estimating the 3-DoF pose (x, y, yaw) by PERCH (Narayanan & Likhachev, 2016b). We also discuss relevant extensions to PERCH. Our discriminative-deliberative framework for 6-Dof pose estimation which relaxes assumptions made by PERCH in 3-Dof is described later in Section 4.6.

3.1. Perception via search for multi-object pose estimation

For estimating the 3-DoF pose (x, y, yaw), PERCH assumes a set of K object instances in the input point-cloud, given the 3D models of N unique objects. PERCH allows the possibility of cases where multiple copies of a particular object instance are present in the scene. The algorithm also assumes that the 6-DoF camera pose is given. The camera pose is required by the algorithm to render 3-Dof poses in the world frame by varying only (x, y, yaw) of every pose. The notations used by PERCH are listed in Table 1. In order to compare PERCH 2.0 with PERCH, we use a similar 3-Dof pose estimation formulation in our work.

3.2. Problem formulation

Given the input point-cloud I, PERCH (Narayanan & Likhachev, 2016b) estimates poses of $O_{1:K}$ objects in the scene, by seeking to find a rendered point-cloud R_j having $j(\leq K)$ objects, such that

Table 1. Notations used in PERCH (Narayanan & Likhachev, 2016b).

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 of R_j 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 $O_{1:j}$
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

every point in I has an associated point in R_j and vice-versa. In other words, PERCH seeks to minimize the following objective:

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

in which $\text{OUTLIER}(p \mid C)$ for a point-cloud C and point p is defined as follows:

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

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. The search tree is referred to as the Monotone Scene Generation tree or MSG (Figure 2).

It is clear that the expansion of each state in the PERCH search tree has a significant computational cost that scales unfavorably 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 with VoxelGrid downsampling, thus obtaining ΔR_j. In order to account for discretization artifacts, local-ICP (Chen & Medioni, 1992) is used to refine the pose. Since the adjusted state may change its

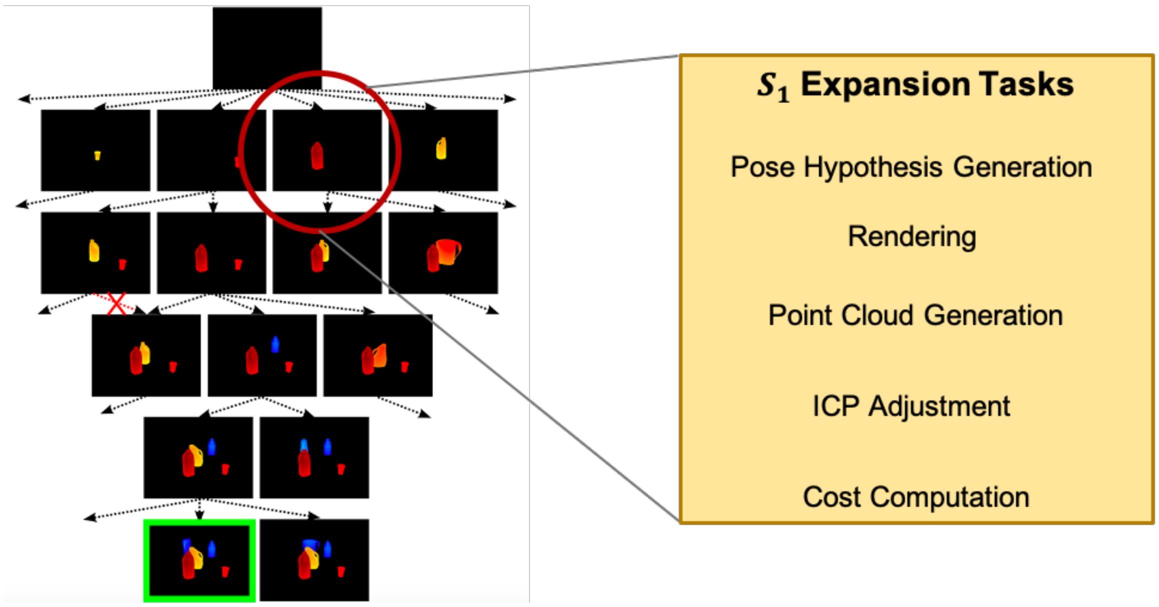


Figure 2. Portion of the Monotone Scene Generation tree constructed by PERCH(Narayanan & Likhachev, 2016b) and corresponding tasks involved in expanding a state S_1.

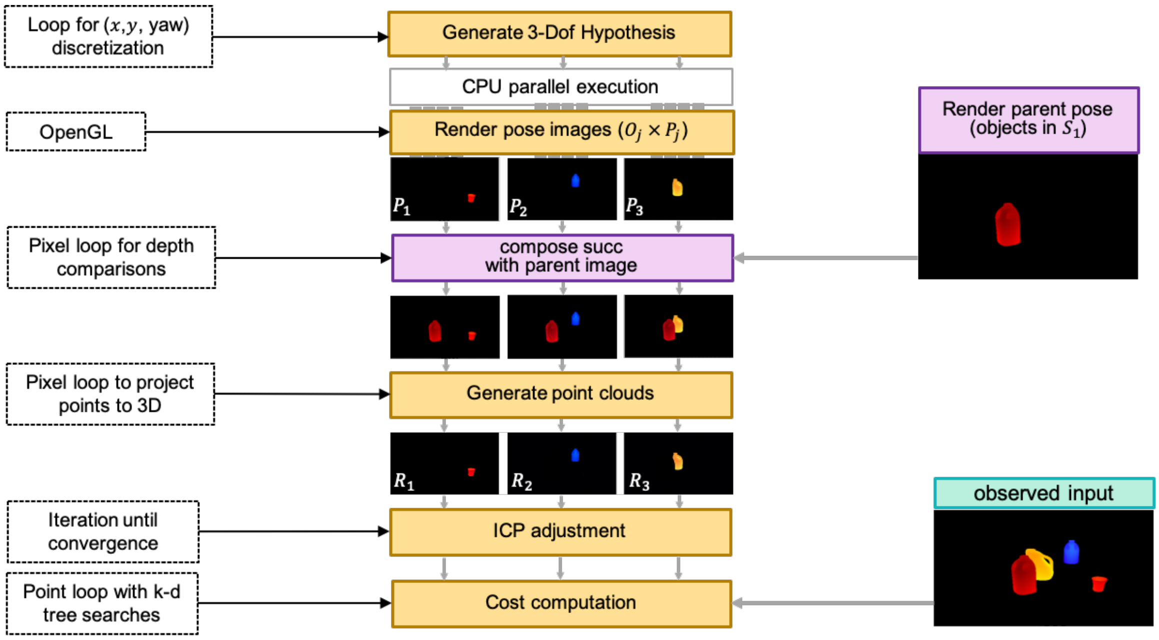

Loop for (x,y, yaw) discretization
OpenGL
Pixel loop for depth comparisons
Pixel loop to project points to 3D
Iteration until convergence
Point loop with k-d tree searches
Generate 3-Dof Hypothesis
CPU parallel execution
Render pose images ($O_j \times P_j$)
P_1
P_2
P_3
compose succ with parent image
Generate point clouds
R_1
R_2
R_3
ICP adjustment
Cost computation
Render parent pose (objects in S_1)
observed input

Figure 3. Expansion of a state S_1 in the PERCH flow on CPU (Narayanan & Likhachev, 2016b).

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 (Bentley, 1975) based nearest neighbor 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 ΔR_j and its nearest neighbor 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 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.

3.3. Extensions to PERCH

In this section we highlight some extensions to PERCH that were proposed in past research and form the inspiration for our work.

3.3.1. Discriminatively-guided Deliberative Perception

The work on D2P (Discriminatively-guided Deliberative Perception (Narayanan & Likhachev, 2016a)) used discriminatively-trained algorithms to guide the tree search, with the objective of reducing the time needed to find a solution. As shown in Figure 4, the input point-cloud is clustered into K components, and the points in each cluster are back-projected to obtain ROIs in the depth image. Then, the ROIs are fed to an R-CNN (Girshick et al., 2014) model trained on the complete object instance database, after appropriate scaling and colorization. Finally, every high-confidence class prediction for an ROI is converted to a heuristic for global search. Let l denote the label associated with a unique object model, B_i the set of ROIs (bounding boxes) in the depth image and $c(l \mid B_i)$ the confidence score for object instance l being present in B_i. For every detection with $c(l \mid B_i) \geq c_{\text{thresh}}$,

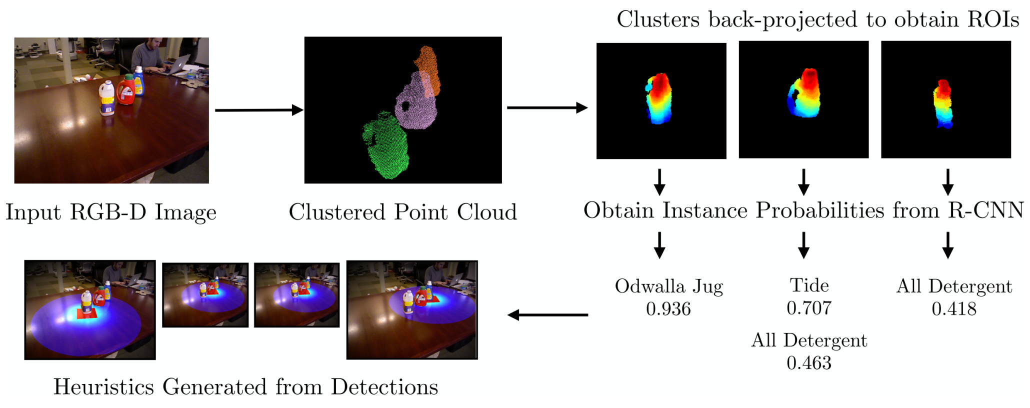

Clusters back-projected to obtain ROIs
Input RGB-D Image
Clustered Point Cloud
Obtain Instance Probabilities from R-CNN
Odwalla Jug 0.936
Tide 0.707
All Detergent 0.463
All Detergent 0.418
Heuristics Generated from Detections

Figure 4. D2P (Narayanan & Likhachev, 2016a) pipeline. In this example, the R-CNN predicts two possible hypotheses for the center ROI, which results in two heuristics being created for that ROI.

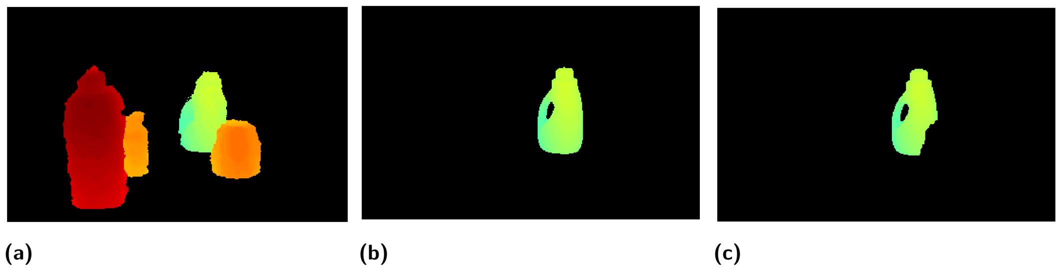


Figure 5. (a) The input point-cloud I (represented as a depth image and pseudo-colored); (b) Rendering R_1 corresponding to one object object O_1; (c) Rendering $R_1 \setminus C_1$, where C_1 corresponds to points in I that occlude R_1 (Narayanan & Likhachev, 2017).

the heuristic is generated as follows:

$$\bar{p} = \text{PROJECTTOSUPPORTPLANE}\left(\text{CENTROID}\left(\{p \mid p \in B_i\}\right)\right) \tag{3}$$

$$h\left(s_j\right) = \begin{cases} \infty & \text{if } \operatorname{id}\left(O_j\right) \neq l \\ 0 & \text{if } \left\|\bar{p} - T\left(O_j\right)\right\|_p \leq r_{\text{detector}} \\ \left\|\bar{p} - T\left(O_j\right)\right\|_p & \text{otherwise} \end{cases} \tag{4}$$

where $\|\cdot\|_p$ is the p-norm and $T(O_j)$ is object O_j's center (assuming all models have been pre-processed such that the z-coordinate of their origins have been set to the height of the supporting surface), ignoring the orientation.

3.3.2. Deliberative perception in clutter

As is evident from previous sections, the Monotone Scene Generation tree formulation is primarily used to account for inter-object occlusions. However, the work on C-Perch (Narayanan & Likhachev, 2017) presented an alternate way to acknowledge inter-object occlusions by marking certain points C_K in the input scene as clutter and using them as extraneous "occluders" while rendering the object of interest in the scene (Figure 5), obtaining the rendered point-cloud $R_K \setminus C_K$ instead of R_K. Here C_K represents the points in the input scene that occlude object $O_{1:K}$. It was shown that this strategy for modelling inter-object occlusions is incredibly useful when models of all objects in the scene are not available and thus the Monotone Scene Generation tree cannot be used to account for the same.

From (Narayanan & Likhachev, 2017), we note the changes to the terms J_o and J_r in Equation 1:

$$\begin{aligned} J_o\left(O_{1:K}\right) &= \sum_{p \in I \cap V_K} \text{OUTLIER}\left(p \mid \left(R_K \setminus C_K\right)\right) \\ J_r\left(O_{1:K}\right) &= \sum_{p \in R_K \setminus C_K} \text{OUTLIER}(p \mid I) \end{aligned} \tag{5}$$

4. Methodology

In this section, we describe our methodology for PERCH 2.0 and how we overcome shortcomings of PERCH as well as some of the related work. In Sections 4.1, 4.2, 4.3 and 4.4, we describe how GPU parallelization of various components is integrated into PERCH 2.0 for runtime improvement. In Section 4.5 we describe the incorporation of RGB data into the PERCH 2.0 cost function, which allows it to distinguish between objects of the same shape. In Section 4.6 we describe the combined discriminative-deliberative framework for 6-Dof pose estimation.

4.1. Parallel scene generation

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 need to be converted to point-clouds with every pixel transformed to its corresponding 3D point using the depth input and camera intrinsic parameters in parallel. Given a depth input D, the following relation can be used to get the (X, Y, Z) location of every pixel (x, y):

$$Z = D(x, y) \tag{6}$$

$$X = \frac{(x - c_U) \times Z}{f_U} \tag{7}$$

$$Y = \frac{(y - c_V) \times Z}{f_V} \tag{8}$$

Here (f_U, f_V) are the camera focal lengths, and (c_U, c_V) are the camera centers. This process can be seen to have a parallelism of $N \times P \times L$ threads where L is the number of points in the rendered image of the pose P.

The Region of Interest or ROI corresponding to the rendered object occupies a relatively small area with respect to the whole image. Even within the ROI, the pixel density is quite high and a direct transformation of every pixel in this region to corresponding (X, Y, Z), would result in a very dense cloud which would in turn limit the scalibility of the downstream tasks. In order to counter this, we introduce a downsampling and masking step that computes a mapping from the pixel space of the rendered images to the reduced 3D point-cloud space while ignoring all pixels outside the ROI. This mapping is then used to compute a downsampled point-cloud for all rendered images in parallel on the GPU. The extent of downsampling can be controlled by a pixel stride parameter that allows the mapping to be sparse or dense as per requirement.

4.2. Parallel Many-to-Many (M2M) GICP

ICP (Chen & Medioni, 1992) is an iterative technique to align a given source point-cloud to a given target point-cloud. An approach of point-to-point non-linear ICP from the PCL library is used by PERCH (Narayanan & Likhachev, 2016b) to align rendered point-clouds to observed clouds and account for discretization artifacts. However as we will show in later sections, this approach is insufficient to deal with the scalability requirements presented by common pose estimation scenarios. Moreover, as we will demonstrate, a point-to-point ICP approach like the one used by PERCH, leads to poor accuracy under high occlusion by converging to the wrong pose. Recent works on GICP (Koide et al., 2021; Segal et al., 2009) 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.

Following Koide et al. (2021), we explain GICP in this section. Let a_i and b_i be two points in source pose and target pose found by a nearest neighbor search and the transform between them is given by T. We assume they are both drawn from Gaussian distributions $a_i \sim \mathcal{N}\left(\hat{a}_i, C_i^A\right)$ and $b_i \sim \mathcal{N}\left(\hat{b}_i, C_i^B\right)$ and the transform error between them is defined as follows:

$$\hat{d}_i = \hat{b}_i - \mathbf{T}\hat{a}_i \tag{9}$$

The error $\hat{d}_i$ can be represented by:

$$\begin{aligned} d_i &\sim \mathcal{N}\left(\hat{b}_i - \mathbf{T}\hat{a}_i, C_i^B + \mathbf{T}C_i^A\mathbf{T}^T\right) &(10)\\ &= \mathcal{N}\left(0, C_i^B + \mathbf{T}C_i^A\mathbf{T}^T\right) &(11)\end{aligned}$$

The GICP objective function is to find the transform T such that:

$$\begin{aligned} \mathbf{T} &= \arg\max_{\mathbf{T}} \sum_i \log\left(p\left(d_i\right)\right) &(12)\\ &= \arg\min_{\mathbf{T}} \sum_i d_i^T \left(C_i^B + \mathbf{T}C_i^A\mathbf{T}^T\right)^{-1} d_i &(13)\end{aligned}$$

The covariance matrices C_i^A and C_i^B are computed from k nearest neighbors in the respective point-clouds. In practice, Equation 12 is solved by using a Gauss Newton optimizer as in Koide et al. (2021). We can see from the outset that performing all the required steps for each candidate pose with the observed scene would be inefficient even if performed in parallel in CPU threads. We propose a modified scalable GPU GICP approach that is shown in Figure 6. The kNN computation required in the flow is described later in Section 4.3.1. In order to compute $J^T J$ and $-J^T L$ from J_i and l_i, cuBLAS batch matrix multiplication is used where each batch represents one pose. The resulting set of equations are solved in parallel using the cuDNN library to obtain ΔT for every pose. This process is repeated iteratively until all poses have converged.

4.3. Parallel objective function evaluation

In this section we describe two approaches that we incorporate in PERCH 2.0 to speed up the evaluation of the objective function. The primary difference in the two approaches is the method used to compute the nearest neighbor distances or kNN which are then used to compute OUTLIERS in Equation 2. In the Experiments section, we present the effects of both approaches on overall runtime.

4.3.1. kNN Approach I

The need to create k-d trees for each successor cloud $\Delta\tilde{R}_j$ and then iteratively computing nearest neighbors in I 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 for computing nearest neighbor distances over N objects, P poses of each object O_j, consisting of L points each in the ICP adjusted point-cloud can be considered as requiring $N \times P \times L \times I$ parallel threads. We use the kNN-GPU library presented in Garcia et al. (2010) to compute the nearest neighbor in the input point-cloud I for each point in every $\Delta\tilde{R}_j$ in order to exploit this parallelism.

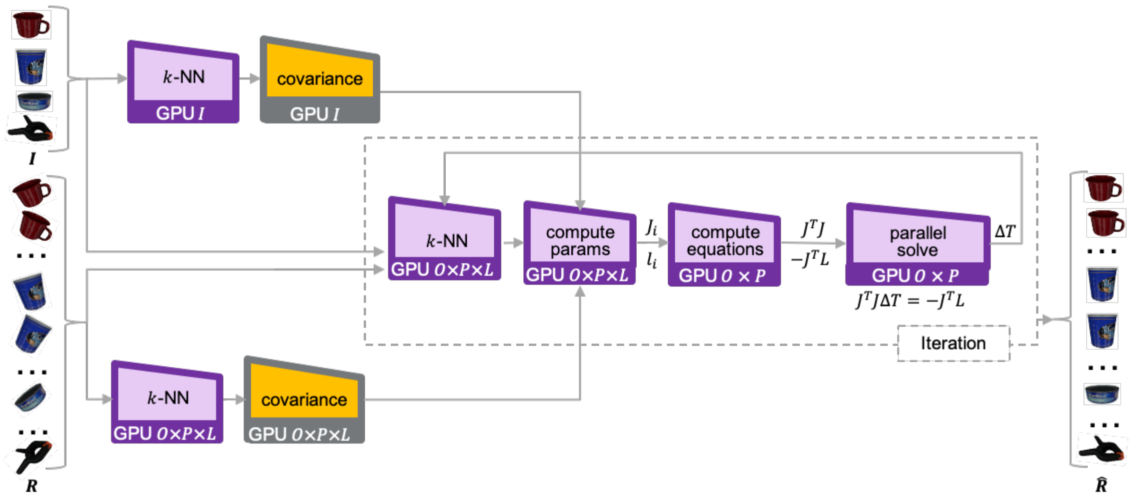


Figure 6. Parallel Many-to-Many GICP flow (O: Number of object instances, P: Number of poses per object instance, L: Number of points in rendered point-cloud of a given pose, I: Input point-cloud, R: Rendered point-clouds, $\hat{R}$: GICP adjusted rendered point-clouds).

4.3.2. kNN Approach II

While Approach I is fully parallel, a closer look at the implementation reveals the need to allocate a large 2D array to store pairwise distances between points in $\Delta\tilde{R_j}$ and I as they are being computed in parallel. This imposes a larger memory requirement on the GPU which could drive up the peak memory usage and limit the overall number of poses that can be evaluated in parallel. Thus we develop another approach 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 execution, we have the nearest neighbors and corresponding distances for every point $\Delta\tilde{R_j}$ to every point in I. Unlike kNN Approach I, the reduced parallelism in this approach eliminates the need for the allocation of a large 2D array.

Once distances are computed by using either approaches, 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 neighbor for a point in the corresponding $\Delta\tilde{R_j}$ in the previous step.

4.4. 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 must determine 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 Narayanan and Likhachev (2016b) that this process is primarily a way to model inter-object occlusions. We build on the strategy used in C-Perch (Narayanan & Likhachev, 2017) 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. Following a strategy similar to Narayanan and Likhachev (2017) for creating $R_K \setminus C_K$ by using the input depth image, we render and compute costs for all successors and find the best one corresponding to the minimum cost for each object in parallel on the GPU. The complete 3-Dof pose estimation pipeline is shown in Figure 7.

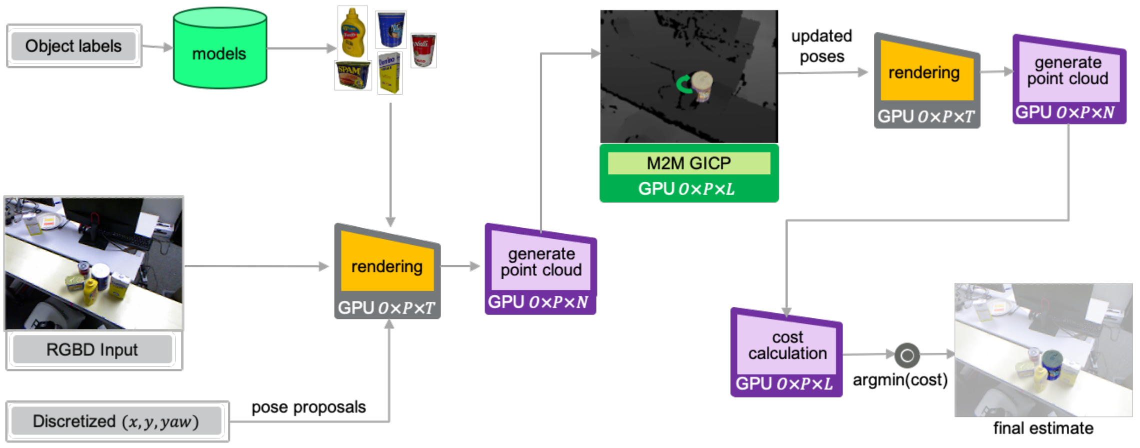

Object labels
models
RGBD Input
Discretized (x, y, yaw)
pose proposals
rendering
GPU $O \times P \times T$
generate point cloud
GPU $O \times P \times N$
M2M GICP
GPU $O \times P \times L$
updated poses
rendering
GPU $O \times P \times T$
generate point cloud
GPU $O \times P \times N$
cost calculation
GPU $O \times P \times L$
argmin(cost)
final estimate

Figure 7. 3-Dof pose estimation flow (O: Number of object instances, P: Number of poses per object instance, T: Number of triangles in an object 3D mesh model, N: Number of pixels in rendered pose image, L: Number of points in given pose point-cloud).

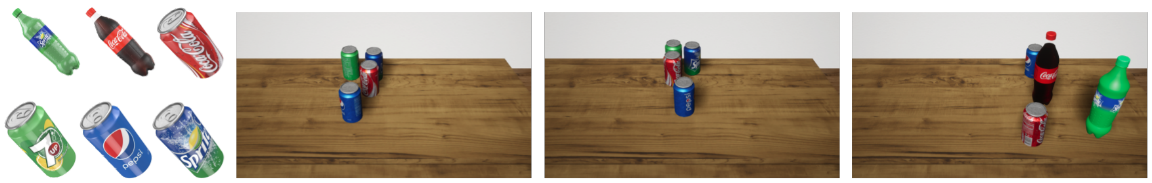


Figure 8. Objects & some sample images from the dataset used for evaluating 3-Dof PERCH 2.0.

4.5. Augmented objective function with RGB

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 8. 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 (Sharma et al., 2005) in the CIELAB color space to perform the comparison between a point in the observed cloud I and the rendered point-cloud $\Delta\tilde{R}_j$ 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 in C, the $\text{OUTLIER}(p \mid C)$ definition in Equation 2 can be re-written as:

$$\text{OUTLIER\_RGBD}(p \mid C) = \begin{cases} 1 & \text{if } \min_{p' \in C} \|p' - p\| > \delta \\ 1 & \text{if } \|p''_c - p_c\|_c > \tau_c \\ & \text{s.t. } \min_{p'' \in C} \|p'' - p\| \leq \delta \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

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(Sharma et al., 2005) 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($p \mid C$), we penalize points for being distant in color space even though they might satisfy the Euclidean constraint in 3D space.

4.6. 6-Dof pose estimation

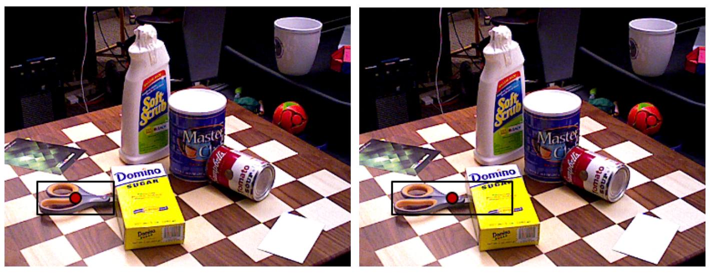
While we presented a way to incorporate RGB data into PERCH 2.0 in Section 4.5, convolutional neural networks (CNNs) have been proven to have extremely high discriminative capabilities on RGB input. We build on the success of CNNs in 2D instance segmentation and it's output to prune a large part of the 6-Dof search space. In our combined discriminative-deliberative framework, we generate a set of pose proposals $H(O_j)$ by combining the 2D object detector with a uniform sampling of the rotation space and then process these proposals using PERCH 2.0 to estimate 6-Dof poses for every object in the scene. Besides extending deliberative pose estimation to 6-Dof, the combined framework avoids several disadvantages of purely discriminative data-driven techniques highlighted in Section 2.1. The integration also relaxes key assumptions made in the PERCH problem formulation, such as, the pre-requisite knowledge labels of objects in scene and the 6-Dof camera pose.

4.6.1. Translation proposal generation

In our approach we generate translation proposals from the 2D bounding box output of the object detection network. Translation proposals denote possible (x, y, z) locations of the center of the object's 3D bounding box. Recent outdoor datasets such as Shao et al. (2018) have focused on annotation of "full" bounding boxes as opposed to only "visible" bounding boxes in the image database. The difference between the two kinds of annotations is highlighted in Figure 9. In our 6-Dof pose estimation framework, we employ "full" bounding box annotation to assist in pose estimation of occluded objects by giving us a more accurate location of the 2D projection of the object's 3D bounding box in the image. With the help of full bounding box and instance segmentation mask, we can generate a set of translation proposals for a detected object O_j as follows:

$$H_{tj} = \langle x_c, y_c, z_i \rangle \qquad \text{where } z_{\min} \leq z_i \leq z_{\max} \tag{15}$$

In the above equation, $\langle x_c, y_c \rangle$ is obtained by back projecting the center of the object's 2D full bounding box into 3D space using the camera's projection matrix. 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.



(a) (b)

Figure 9. Different kinds of bounding box annotations. (a) Visible bounding box; (b) Full bounding box.

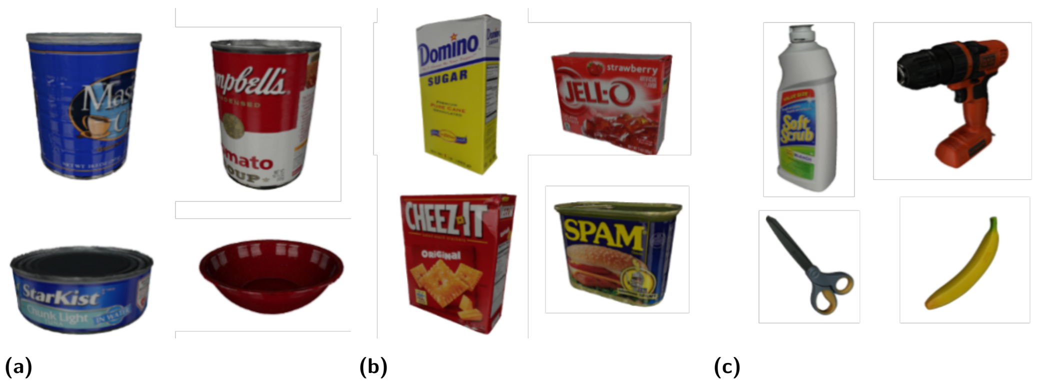


Figure 10. (a) Objects from the YCB object database (Calli et al., 2015) with rotational symmetry about one axis; (b) Objects from the YCB object database (Calli et al., 2015) with 180 degree semi-symmetry about 2 axis; (c) Objects from the YCB object database (Calli et al., 2015) with no symmetry about any axis.

4.6.2. Rotation proposal generation

In our approach, we uniformly sample the rotation space by representing all possible rotations as a set of viewpoints and in-plane rotation angles (Kehl et al., 2017; Sundermeyer et al., 2018). We equidistantly-sample M viewpoints v from unit sphere and N in-plane rotation angles θ from $[0, 2\pi]$ and combine each with the other to generate $M \times N$ possible set of rotation proposals for object O_j:

$$H_{rj} = \langle v_i, \theta_k \rangle \qquad \text{where } 1 \leq i \leq M \text{ and } 1 \leq k \leq N \tag{16}$$

An advantage of sampling rotations using viewpoints and in-plane rotations is that it allows for explicit handling of object symmetry while ensuring that all object orientations have been considered. This is in contrast to CNN based pose estimation methods where learning object symmetries is left to the neural network. For an object that exhibits rotational symmetric about one axis such as the can shown in Figure 10, the sampling of in-plane rotation can be skipped, significantly reducing the number of proposals to be considered. Similarly for an object such as a sugar box which is semi-symmetric or exhibits indistinguishable views if rotated along the axis by 180 degrees, only half of the viewpoint sphere needs to be sampled.

4.6.3. 6-Dof pose estimation pipeline

A pictorial representation of the entire 6-Dof pose estimation pipeline can be seen in Figure 11. The input RGB image is passed through a MaskRCNN (He et al., 2017) 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. 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 our parallel M2M GICP approach. We note that instance segmentation fits neatly into our parallel GICP framework, allowing us to align multiple source poses to multiple target poses in parallel. In addition, we use instance segmentation labels to speed up k-NN II by computing nearest neighbors only for matching labels in rendered and observed point-clouds.

Finally, we render and generate point-clouds for the adjusted poses and compute the cost of each pose proposal in parallel.

Instead of explicitly computing points within the object volume $V(O_j)$ for calculation of observed cost component J_o, we directly use the pixel-wise segmentation labels to determine the set of observed points belonging to a given object.

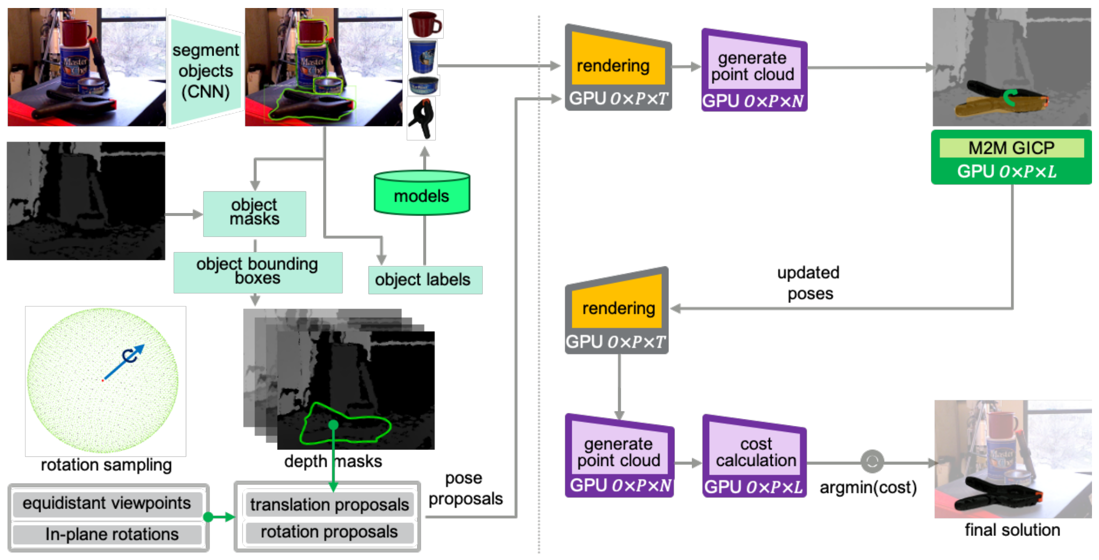


Figure 11. 6-Dof Pose Estimation Pipeline (O: Number of object instances, P: Number of poses per object instance, T: Number of triangles in an object 3D mesh model, N: Number of pixels in a rendered pose image, L: Number of points in given pose point-cloud).

5. Experimental results

5.1. 6-Dof tabletop manipulation

In this section we evaluate PERCH 2.0 on 6-Dof pose estimation and compare it against state-of-the-art, end-to-end, learning-based methods on the YCB-Video Dataset (Xiang et al., 2018).

5.1.1. *External baselines*

In order to evaluate the performance of PERCH 2.0 for 6-Dof pose estimation, we compare our results with DenseFusion (Wang et al., 2019) and PoseCNN + ICP (Xiang et al., 2018) on objects from the YCB-Video Dataset (Xiang et al., 2018). The results are computed for the 2,949 keyframes used for testing in prior works.

5.1.2. *Ablation baselines*

In order to understand the impact of various components in PERCH 2.0 on the overall pose estimation accuracy, we use the following PERCH 2.0 baselines:

- *PERCH 2.0 A (PoseCNN Mask + PCL ICP + kNN I)*: Here we directly instance segmentation masks published online by PoseCNN[1] in conjunction with *PERCH 2.0 (PCL ICP + kNN I)*. PCL ICP is the same non linear ICP algorithm used by PERCH. Since the mask is directly used, we do not have the full bounding boxes available. The bounding box is computed from the mask provided and the center of bounding box is used in PERCH 2.0. We note that this is the visible bounding box as opposed to the full bounding box that our method seeks to use.
- *PERCH 2.0 B (PoseCNN Mask + M2M GICP + kNN II)*: Here, we directly use the PoseCNN masks in conjunction with *PERCH 2.0 (M2M GICP + kNN II)* which uses the our parallel many-to-many GICP approach (Section 4.2) and kNN II (Section 4.3.2).
- *PERCH 2.0 C (MaskRCNN + M2M GICP + kNN II)*: Here we use a MaskRCNN model that outputs instance segmentation masks and full bounding boxes in conjunction with *PERCH 2.0*

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

(M2M GICP + kNN II). This is our recommended framework for 6-Dof pose estimation. The MaskRCNN implementation is described in Section 5.1.3 below.

For a better understanding of the impact of the GPU-based parallel M2M GICP approach and kNN II on runtime, we use two variants of PERCH 2.0:

- *PERCH 2.0 C (kNN I + CPU GICP)* which uses kNN I and the publicly available CPU parallelized version of GICP (Koide et al., 2021).
- *PERCH 2.0 D (kNN II + M2M GICP)* which uses our parallel many-to-many GICP approach (Section 4.2) and kNN II (Section 4.3.2)

For running the above PERCH 2.0 variants, we use a machine with 16GB P100 NVidia GPU and 32 CPU cores. For rotation proposals, we sample 80 viewpoints from the unit sphere and combine it with 3 in-plane rotation angles (for unsymmetrical objects). Translation proposals are created by sampling object centers at a 1 cm threshold. A pixel stride of 8 is used to create downsampled point-clouds from RGBD inputs.

5.1.3. *MaskRCNN implementation details*

In order to train a CNN for instance segmentation and full bounding box detection, we use a Mask RCNN (He et al., 2017) model to train on RGB images, segmentation and class labels from the 80 training videos provided in the YCB-Video Dataset. Our implementation is based on Massa and Girshick (2018). However since the YCB-Video dataset doesn't contain annotation for full bounding boxes, we use the ground-truth 6-Dof pose and project it onto the image to obtain the full bounding box. We note that though we are using the 6-Dof pose to obtain the full bounding box, this could instead be done through crowdsourced human annotation as done for the CrowdHuman (Shao et al., 2018) dataset, eliminating the need to collect 6-Dof pose-annotations. The training is performed on 4 NVidia V100 GPUs.

Metrics. We use two metrics to compare our 6-Dof pose estimation results—area under ADD-S threshold curve (ADD-S AUC < 0.1 m) and the percentage of poses with ADD-S < 2 cm. The results are shown in Table 2. For understanding the accuracy of our instance segmentation and bounding box regression, we use the average precision or the AP metric (He et al., 2017). The results are shown in Table 3.

Accuracy. The results in Table 2 show that without using any ground-truth pose-annotation, the variant *PERCH 2.0 B* outperforms all external baselines on both AUC (< 0.1 m) and ADD-S < 2 cm metrics even though it uses the same PoseCNN Mask. We observe that the variant *PERCH 2.0 C* that uses full bounding boxes further improves on the accuracy and performs well across objects of varying shape, size, texture and symmetry, estimating 99.29% of the poses within the 2 cm ADD-S error and hence within the tolerance limit of most robot grippers. However unlike DenseFusion (Wang et al., 2019) and PoseCNN + ICP (Xiang et al., 2018), no pose estimation network on images from the 80 training videos and 80,000 synthetic images were trained. The results in Table 3 show that our MaskRCNN model is able to accurately estimate full bounding boxes and instance segmentation masks for objects of different shapes and sizes. On closer inspection, we could see that the reason for the low AP for 051_large_clamp and 052_extra_large_clamp is because the model often confuses between them due their very similar appearance with the only difference being their size.

Robustness. A comparison between *PERCH 2.0 B* with *PERCH 2.0 A* shows the improvement that can be achieved with the help of GICP alone. Further, the comparison between *PERCH 2.0 C* and *PERCH 2.0 B* shows the improvement that can be achieved with full bounding boxes. Specifically, a significant jump in accuracy metrics is seen for highly occluded objects such as the 003_cracker_box, 024_bowl, and 021_bleach_cleanser.

Table 2. Area under accuracy-threshold curves for 6-Dof pose estimation on objects from the YCB-Video Dataset (Xiang et al., 2018).

Objects	**PoseCNN + ICP**		**DenseFusion (Iterative)**		**PERCH 2.0-A (PoseCNN Mask + PCL ICP + *k*NN-I)**		**PERCH 2.0-B (PoseCNN Mask + M2M GICP + *k*NN-II)**		**PERCH 2.0-C (MaskRCNN Mask + M2M GICP + *k*NN-II)**	
	AUC	**<2 cm**	**AUC**	**<2 cm**	**AUC**	**<2 cm**	**AUC**	**<2 cm**	**AUC**	**<2 cm**
002_master_chef_can	95.80	100.00	**96.40**	100.00	95.3	99.86	96.06	100.00	96.25	**100.00**
003_cracker_box	92.70	91.60	**95.50**	99.50	89.14	96.69	93.54	97.81	94.69	**99.65**
004_sugar_box	**98.20**	100.00	97.50	**100.0**	98.64	98.64	95.86	99.66	96.11	99.58
005_tomato_soup_can	94.50	96.90	94.60	96.9	95.62	100	97.26	99.77	**97.30**	**100.00**
006_mustard_bottle	**98.60**	100.00	97.20	100.0	93.99	100	97.51	100.00	97.42	**100.00**
007_tuna_fish_can	**97.10**	100.00	96.60	100	93.59	100	95.50	99.91	95.97	**100.00**
008_pudding_box	**97.90**	100.00	96.50	**100**	92.32	97.7	93.04	94.03	93.55	99.53
009_gelatin_box	**98.80**	100.00	98.10	100	92.46	100	96.77	100.00	96.56	**100.00**
010_potted_meat_can	92.70	93.60	91.30	93.1	93.64	97.99	95.13	97.82	**95.45**	**99.72**
011_banana	**97.10**	99.70	96.60	**100**	94.93	99.73	96.53	99.74	96.88	99.74
019_pitcher_base	**97.80**	100.00	97.10	100	92.4	100	92.37	100.00	92.11	**100.00**
021_bleach_cleanser	**96.90**	99.40	95.80	100	90.81	99.03	93.39	96.99	95.25	**100.00**
024 bowl	81.00	54.90	88.20	98.8	92.55	95.56	93.42	97.04	**97.22**	**100.00**
025_mug	95.00	99.80	**97.10**	100	95.66	100	96.96	100.00	96.96	**100.00**
035_power_drill	**98.20**	99.60	96.00	98.7	92.94	**100**	96.10	99.91	95.72	99.72
036_wood_block	87.60	80.20	89.70	**94.60**	90.07	94.6	90.31	90.08	**91.58**	93.61
037_scissors	91.70	95.60	95.20	100	91.11	96.65	95.11	100.00	**96.49**	**100.00**
040_large_marker	97.20	99.70	97.50	100	96.11	100	97.56	99.85	**97.78**	**100.00**
051_large_clamp	75.20	74.90	72.90	79.20	84.16	71.19	72.25	77.06	**92.41**	**97.99**
052_extra_large_clamp	64.40	48.80	69.80	**76.3**	80.66	71.54	86.12	82.58	**88.54**	**90.24**
061_foam_brick	**97.20**	100.00	92.50	100.0	94.48	100	95.89	100.00	95.72	**100.00**
All Objects	93.00	93.20	93.10	96.8	92.10	96.15	94.56	98.00	**95.48**	**99.29**

Runtime. From Table 4, we can observe that *PERCH 2.0 C* on 6-Dof poses takes an average of 75.43 s to estimate poses for all objects in the scene. A closer inspection reveals that out of the 75.4 s, an average of 88.9% time is spent on refining poses through ICP, while other parts of the pipeline take an average of only 8.4 s to run for all objects and poses in the scene. In contrast, *PERCH 2.0 B* takes only 7.6 s on average to estimate poses for all objects in the scene. It achieves a ∼10X improvement over *PERCH 2.0 A*, highlighting the importance of parallel GICP and *k*NN 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.

Qualitative analysis. A few test scenes from the YCB-Video Dataset along with corresponding bounding box, instance segmentation and 6-Dof pose predictions are shown in Figure 12. From the first scene, we can observe that despite severe occlusion of the 037_scissors object, our Mask RCNN model is able to accurately localize the full 2D bounding box and hence the object center in the image. The predicted poses in scenes show that 6-Dof poses of objects of various shapes, sizes, symmetries with varying levels of occlusion are predicted accurately. Figure 13 shows a scenario where pose estimates from PERCH 2.0 could be inaccurate. The inaccuracy in this case is caused by depth holes in the reflective parts of the 007_tuna_fish_can, resulting in very few points in the corresponding point-cloud and hence an inaccurate pose estimate. Similarly Figure 14 depicts a scenario where predicted pose estimates could be ambiguous. The ambiguity is caused by a reduction in density of the observed point-clouds of 019_pitcher_base and 035_power_drill at certain angles, which in turn results in multiple poses matching the observation during M2M GICP and cost computation.

Table 3. Evaluation of bounding box regression and instance segmentation on the YCB-Video Dataset (Xiang et al., 2018) using Mask-RCNN.

Objects	**Bounding Box Regression**						**Instance Segmentation**					
	AP	**AP50**	**AP75**	**APs**	**APm**	**APl**	**AP**	**AP50**	**AP75**	**APs**	**APm**	**APl**
002_master_chef_can	0.825	1.000	1.000	−1.000	−1.000	0.825	0.711	1.000	0.854	−1.000	−1.000	0.744
003_cracker_box	0.785	1.000	0.990	−1.000	−1.000	0.785	0.693	1.000	0.781	−1.000	−1.000	0.693
004_sugar_box	0.849	1.000	0.990	−1.000	−1.000	0.849	0.837	1.000	1.000	−1.000	−1.000	0.837
005_tomato_soup_can	0.794	0.980	0.970	0.042	0.804	0.805	0.772	0.970	0.948	0.000	0.730	0.803
006_mustard_bottle	0.890	1.000	1.000	−1.000	−1.000	0.890	0.851	1.000	1.000	−1.000	−1.000	0.851
007_tuna_fish_can	0.824	1.000	1.000	−1.000	0.828	0.822	0.816	1.000	0.989	−1.000	0.810	0.839
008_pudding_box	0.720	0.958	0.938	−1.000	0.745	0.728	0.548	0.958	0.548	−1.000	0.495	0.618
009_gelatin_box	0.823	1.000	1.000	−1.000	−1.000	0.823	0.846	1.000	0.970	−1.000	−1.000	0.846
010_potted_meat_can	0.720	0.938	0.842	−1.000	0.565	0.843	0.759	0.990	0.882	−1.000	0.657	0.849
011_banana	0.802	1.000	0.975	−1.000	0.822	0.814	0.807	1.000	1.000	−1.000	0.807	0.810
019_pitcher_base	0.908	1.000	1.000	−1.000	−1.000	0.908	0.931	1.000	1.000	−1.000	−1.000	0.931
021_bleach_cleanser	0.773	1.000	0.956	−1.000	0.608	0.785	0.811	1.000	0.988	−1.000	0.684	0.821
024_bowl	0.726	1.000	0.867	−1.000	0.451	0.731	0.812	0.987	0.969	−1.000	0.026	0.828
025_mug	0.817	1.000	1.000	−1.000	0.854	0.795	0.665	1.000	0.861	−1.000	0.656	0.676
035_power_drill	0.724	1.000	0.939	−1.000	−1.000	0.724	0.629	1.000	0.721	−1.000	−1.000	0.631
036_wood_block	0.557	0.693	0.679	−1.000	−1.000	0.557	0.483	0.692	0.581	−1.000	−1.000	0.485
037_scissors	0.368	0.748	0.309	−1.000	0.416	−1.000	0.471	0.798	0.606	−1.000	0.477	−1.000
040_large_marker	0.717	1.000	0.906	−1.000	0.727	−1.000	0.497	0.984	0.441	−1.000	0.497	−1.000
051_large_clamp	0.456	0.658	0.578	−1.000	0.539	0.314	0.316	0.653	0.160	−1.000	0.303	0.397
052_extra_large_clamp	0.513	0.774	0.725	−1.000	0.601	0.538	0.437	0.774	0.492	−1.000	0.095	0.572
061_foam_brick	0.798	1.000	0.948	−1.000	0.804	0.848	0.752	1.000	0.969	−1.000	0.742	0.867
Mean AP	0.733	0.940	0.886	0.042	0.674	0.757	0.688	0.943	0.798	0.000	0.537	0.742

Table 4. Evaluation of runtime on YCB-Video Dataset.

Method	Runtime (s)
PoseCNN + ICP	10
DenseFusion (Iterative)	0.06
PERCH 2.0 C (kNN I + CPU GICP)	75.43
PERCH 2.0 D (kNN II + M2M GICP)	7.6

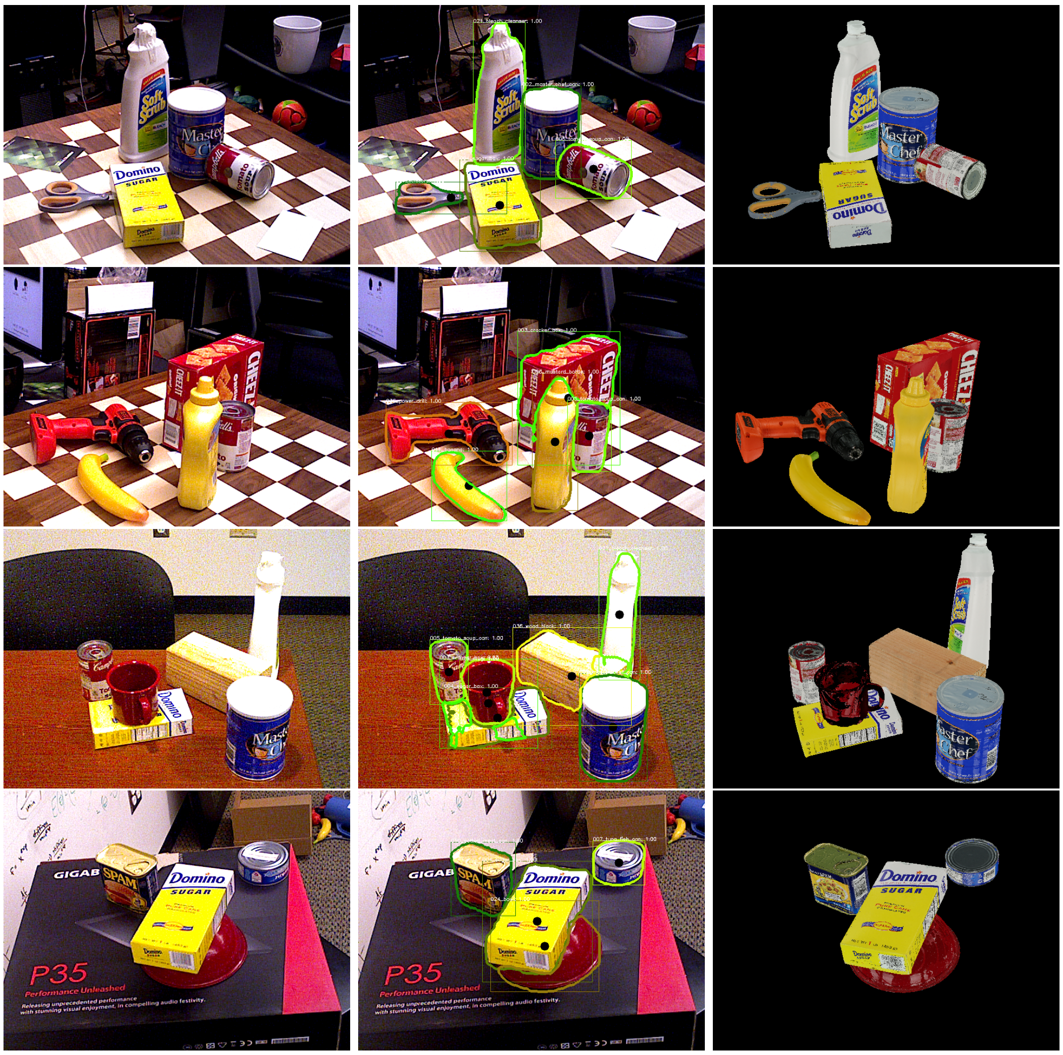
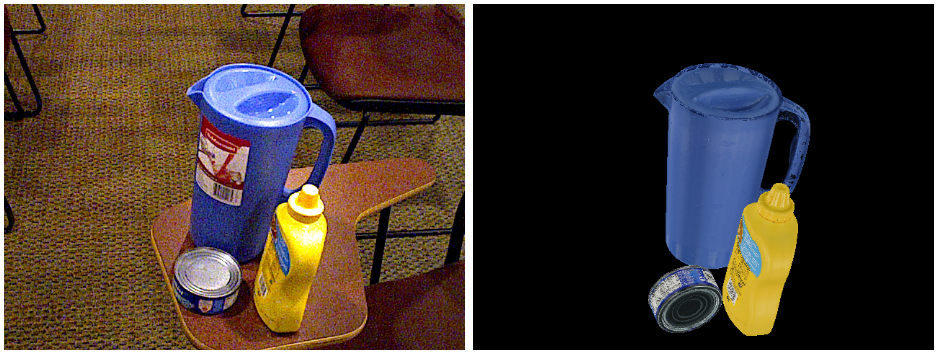

(a) Input scene **(b)** 2D object detection **(c)** Predicted 6-Dof poses

Figure 12. Results from the YCB-Video Dataset.

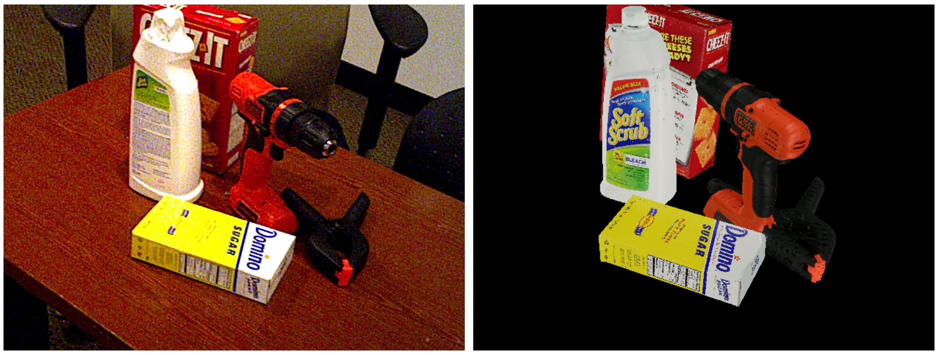
5.2. 3-Dof pose estimation

While the previous section evaluated PERCH 2.0 against other publicly available methods, it is inadequate for evaluation against PERCH, since a 6-Dof pose estimation approach based on the hierarchical PERCH would be impractical owing to its high run time. Thus, in this section, we describe several experiments on 3-Dof pose estimation to evaluate the performance of PERCH 2.0 in direct comparison with PERCH. Due to the lack of availability of 3-Dof pose estimation datasets,



(a) Input scene **(b)** Reconstructed scene from predicted poses

Figure 13. Edge case from the YCB-Video Dataset: Inaccurate pose for 007_tuna_fish_can.



(a) Input scene **(b)** Reconstructed scene from predicted poses

Figure 14. Edge case from the YCB-Video Dataset: Pose ambiguity for 019_pitcher_base and 035_power_drill.

we construct our own synthetic or real world datasets for these experiments with ground-truth pose-annotation.

5.2.1. Tabletop object manipulation

Dataset. Experiments in Narayanan and Likhachev (2016b) have shown 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 database exists in literature that consists of depth and RGB 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 and depth images using NVidia NDDS (To et al., 2018) plugin for Unreal Engine 4 (objects shown in Figure 8). 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 generates images with more realistic lighting conditions and inter-object occlusion.

Baselines. We use the following baselines and PERCH 2.0 variants for this experiment:

- DOPE (Tremblay et al., 2018) + ICP: DOPE is a RGB 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.

- The Brute Force ICP (BF-ICP) baseline presented in Narayanan and Likhachev (2016b) is also used for comparison. Here the object's model is transformed by the same candidate poses considered by PERCH and PERCH 2.0, and then local ICP alignment is performed and the pose with the best ICP fitness score is selected.
- PERCH 2.0-A which does not use the input depth data to mark occluded points in the rendered scenes.
- PERCH 2.0-B which does not use full parallelization like PERCH 2.0, but instead uses a tree search formulation similar to PERCH.
- PERCH 2.0-C which uses full parallelization, kNN I and the same PCL ICP used by PERCH.

For training DOPE, we construct a training dataset of 12K images containing each of the selected 6 objects using NDDS (To et al., 2018). 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. For inference of DOPE and detection using PERCH and PERCH 2.0, an 8 core Intel i7-6700 CPU with an NVidia GeForce GTX 1070 8GB GPU was 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 color distance threshold τ_c is set to 12.5.

Metrics. We use the ADD-S (Hinterstoisser et al., 2013; Xiang et al., 2018) 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 5. We also compute ADD-S < 1 cm, which denotes the percentage of poses with less than 1 cm ADD-S error.

Accuracy. PERCH 2.0-C achieves the best performance among all variants with 100% of poses below ADD-S 1 cm error. It can also be noted that PERCH 2.0-C and DOPE + ICP outperform PERCH and BF-ICP. This shows that PERCH 2.0 and DOPE are able to utilize the RGB information present in the object model and observed scene and closer inspection reveals that these methods do not 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

Table 5. Evaluation of 3-Dof pose estimation for similar shape objects.

Objects	**BF-ICP**		**PERCH**		**DOPE + ICP**		**PERCH 2.0-A (W/O Occluder Marking)**		**PERCH 2.0-B (W/O Full Parallelization)**		**PERCH 2.0-C (ICP + kNN I)**	
	AUC	**<1 cm**	**AUC**	**<1 cm**	**AUC**	**<1 cm**	**AUC**	**<1 cm**	**AUC**	**<1 cm**	**AUC**	**<1 cm**
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	

can, 7up can). PERCH 2.0-C also handles occlusions more effectively as compared to DOPE and PERCH 2.0-A, which is exhibited in its better performance as compared to both.

Runtime. From Table 5 it is clear that we are able to achieve a nearly two order of magnitude improvement in runtime (∼100X) with PERCH 2.0-C over PERCH. 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 (Narayanan & Likhachev, 2016b). 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.

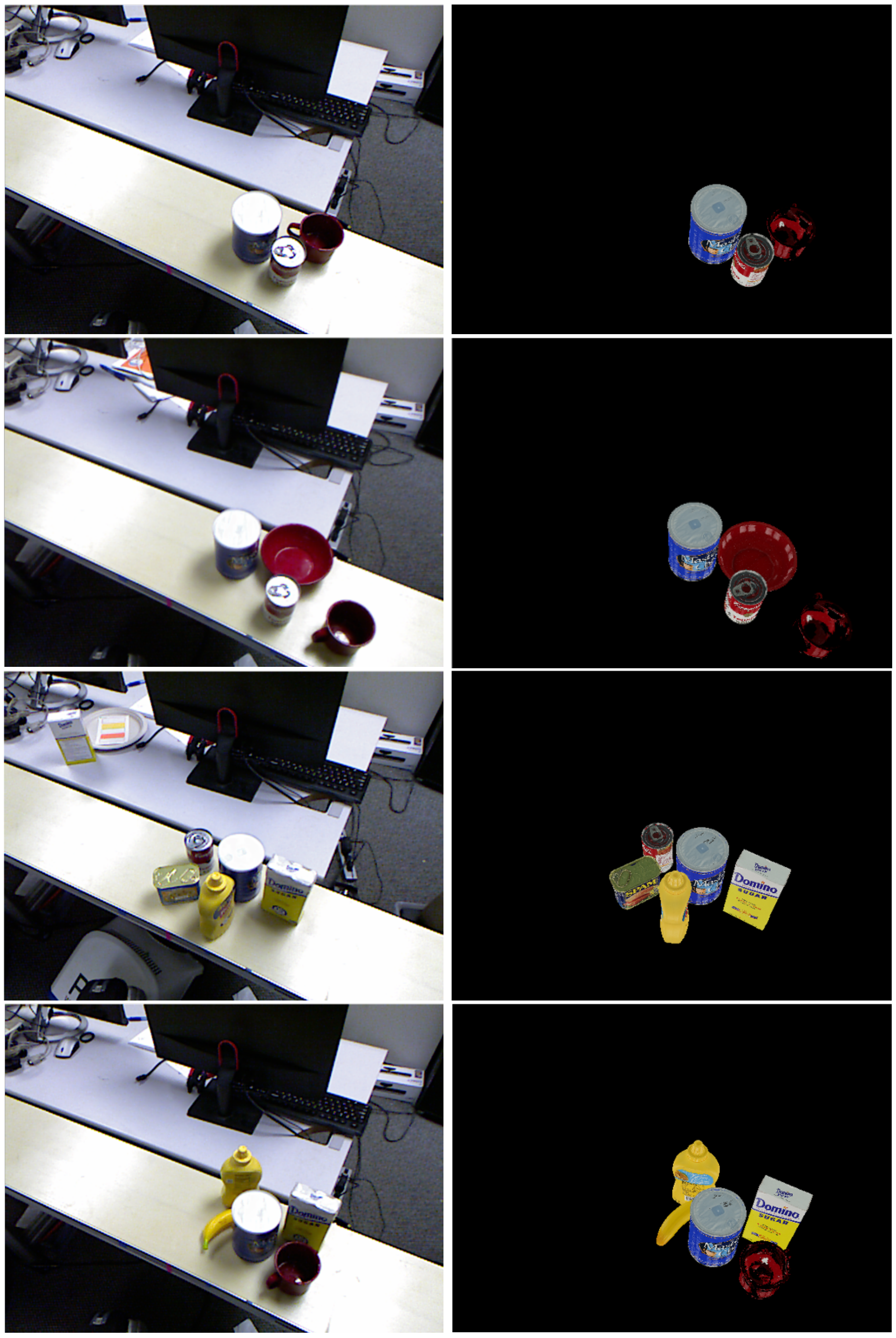
Qualitative results. We analyzed the performance of PERCH 2.0 with several scenes captured on 2 robot systems—a PR2 mobile manipulator and a Ubtech Walker robot. The results are shown in Figure 15 and Figure 16. The results on PR2 in Figure 15 demonstrate that the pose estimation accuracy and runtime transfer well to the real world scenes that consist of multiple objects with inter-object occlusion. The results on the Walker robot in Figure 16 show that the RGB aware cost function is capable of distinguishing objects of similar shape but different appearance in the real world. We also note that PERCH 2.0 was used in Saxena et al. (2021) to perform real world manipulation based on the 3-Dof poses predicted by the algorithm.

5.2.2. Conveyor object manipulation

Task. Recent research has explored mobile manipulators being deployed for the task of pick-and-drop of moving objects on a conveyor (Cowley et al., 2013; Islam et al., 2021; Menon et al., 2014). As argued in Cowley et al. (2013), a fast and accurate perception system is needed for robust pick-and-place of moving objects. Further, authors in Islam et al. (2021) showed that an accurate and early pose estimate can be utilized by motion planners to pick up objects moving at high speeds. However due to high runtime, deliberative pose estimation methods like PERCH are unable to be deployed for tasks such as these while data-drive techniques like Wang et al. (2019) would have to be trained for every unique object and at different distances from the camera. In this section, we demonstrate that due to runtime enhancement, PERCH 2.0 can successfully estimate the 3-Dof poses of objects moving on a conveyor with high accuracy and speed.

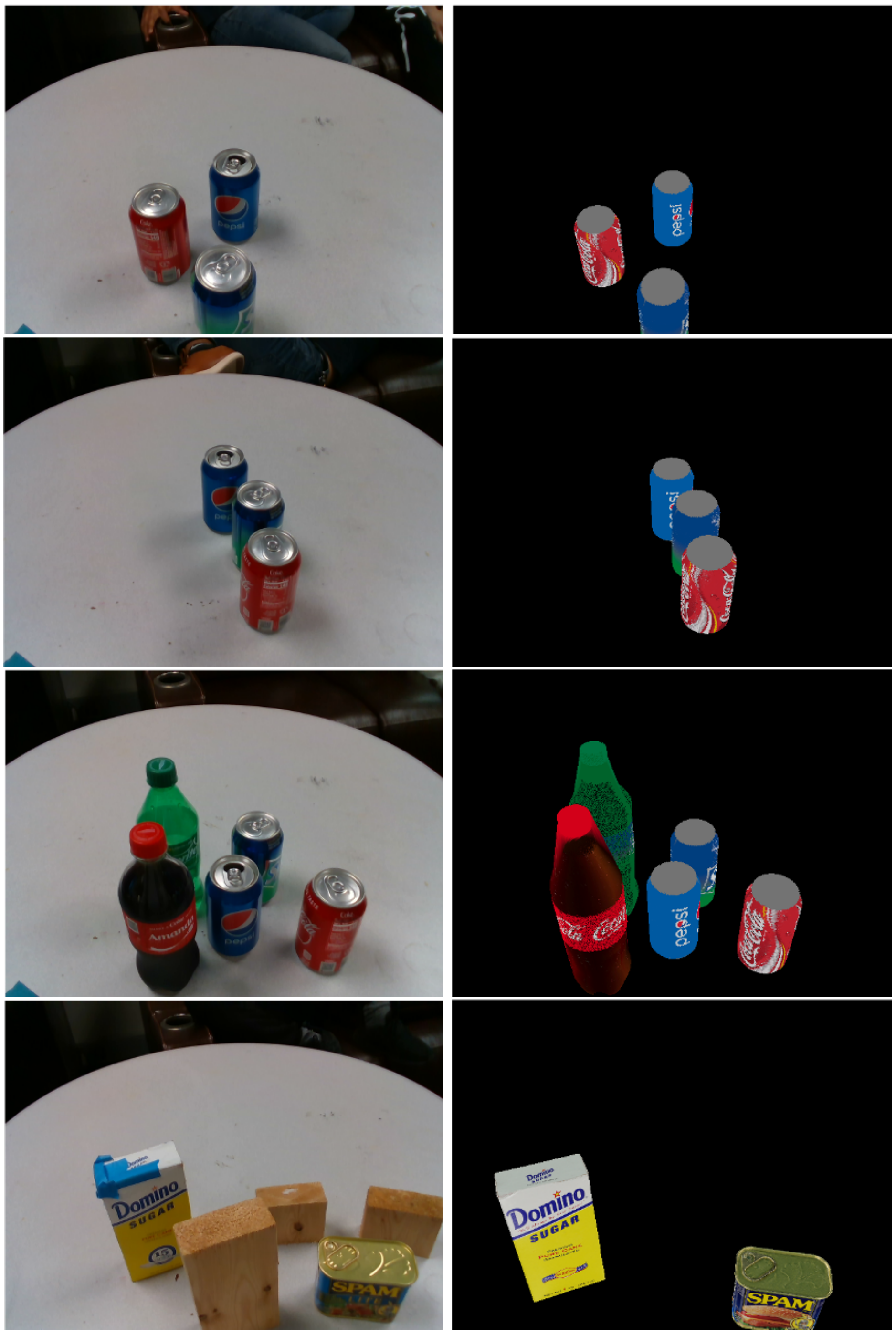
Dataset. Since there is no publicly available database that consists of sequences of moving objects with known 3D models and varying 3-Dof poses, we constructed a database by collecting 10 videos of 4 YCB objects moving along the conveyor. The objects are shown in Figure 17. The 6-Dof camera pose was also captured along with intrinsic camera parameters. In order to annotate the ground-truth 3-Dof pose, we first split the recorded videos into RGB and depth images. For each captured frame, we convert the corresponding 3D point-cloud to the frame of the robot base and then filter out all points except those belonging to the object of interest using pass-through filters in the PCL library (Rusu & Cousins, 2011). Next we manually annotate the 3-Dof pose of the object in the first captured frame. For every successive frame, we use ICP alignment from PCL to annotate the pose using the annotation from the previous frame as the initial pose estimate. In this manner, we generate 961 RGB and depth images with annotated 3-Dof poses.

Baselines. For this experiment we use PERCH 2.0 with the same ICP as PERCH and kNN I (Section 4.3.1). Similar to the previous experiment, our objective here is to examine the adaptability of deliberative pose estimation methods like PERCH to this task, as well as to test the runtime improvement offered by PERCH 2.0. Consequently, we evaluate both algorithms on the generated conveyor dataset. We evaluate both algorithms against the BF-ICP baseline as before. We note that the BF-ICP approach was used in Islam et al. (2021) to perform pose estimation for real-time grasping of objects off a conveyor. For both PERCH and PERCH 2.0, we use a translation discretization of



(a) Input Scene (b) Reconstructed scene from predicted poses

Figure 15. A few qualitative 3-Dof pose estimation results on PR2 for runtime evaluation (Average runtime: 2.45 s).



(a) Input Scene **(b)** Reconstructed scene from predicted poses

Figure 16. A few qualitative 3-Dof pose estimation results on Walker Robot for RGBD cost function evaluation.

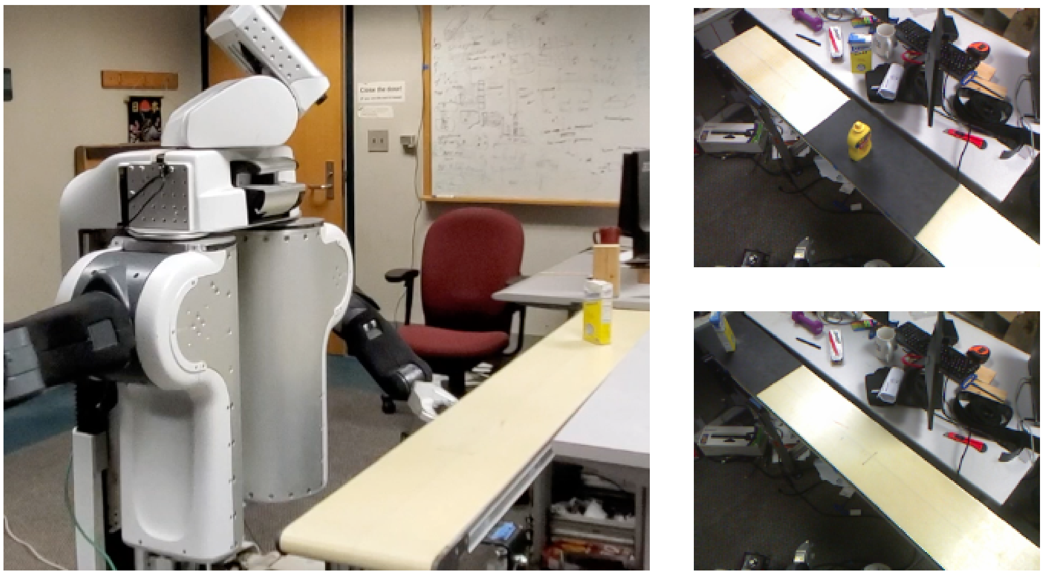


Figure 17. Left: PR2 robot setup for data collection, Right: A few extracted frames from the conveyor dataset.

Table 6. Evaluation of 3-Dof pose estimation of objects on a conveyor.

Objects	**BF-ICP**		**PERCH**		**PERCH 2.0 (ICP + *k*NN I)**		**Database**
	AUC	**<2 cm**	**AUC**	**<2 cm**	**AUC**	**<2 cm**	**Image Count**
035_power_drill	91.38	100.000	91.60	99.785	91.37	99.57	465
006_mustard_bottle	85.27	82.278	91.93	98.101	91.70	98.73	158
004_sugar_box	87.85	96.842	91.20	99.648	91.06	99.30	284
005_tomato_soup_can	92.25	92.593	94.18	100.000	94.85	100.00	54
All Objects	89.29	95.738	**91.61**	**99.479**	91.47	99.38	961
Mean Runtime (seconds)	8.065		25.938		**0.619**		

0.08 m and a yaw discretization of 22.5 degrees. The sensor resolution δ is set to 0.02 m. For running PERCH 2.0 and PERCH, a machine with a GTX 1070 8GB GPU with 8 CPU cores is used.

Metrics. We use three metrics to compare our results—area under ADD-S threshold curve for every object (ADD-S AUC < 0.1 m), the percentage of poses with ADD-S < 2 cm, and the mean ADD-S error. The results are shown in Table 6.

Accuracy. PERCH 2.0 and PERCH achieve comparable performance on the dataset with more than 99% of the poses within the 2 cm ADD-S error limit. The slight difference in performance can be attributed to differences in downsampling used by the two methods. While PERCH uses Voxel downsampling in 3D point-cloud space, PERCH 2.0 samples directly in the image space using a stride to skip pixels. Both methods perform significantly better than BF-ICP, again highlighting the importance of rendering in achieving high accuracy. The improvement in accuracy when compared to BF-ICP suggests that PERCH 2.0 is a viable candidate for performing pose estimation for the conveyor picking task since BF-ICP was used in Islam et al. (2021).

Runtime. We can observe from Table 6 that PERCH 2.0 is significantly faster than PERCH and offers a $\sim$43X speedup. In order to understand the impact of runtime on the given task, we note that

Table 7. Maximum number of possible poses processed for given conveyor speed.

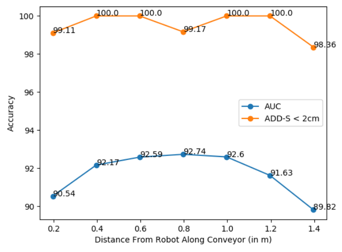
Conveyor Speed (m/s)	Number of Poses Processed	
	PERCH	PERCH 2.0
0.025	2	90
0.050	1	45
0.100	1	23
0.150	0	15
0.200	0	11
0.250	0	9
0.300	0	8
0.350	0	6

Table 8. Variation of accuracy metrics with distance from the robot along the conveyor.

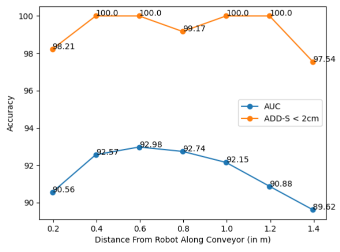
Distance From Robot (m)	BF-ICP		PERCH		PERCH 2.0	
	AUC	$<2\,\mathrm{cm}$	AUC	$<2\,\mathrm{cm}$	AUC	$<2\,\mathrm{cm}$
0.0–0.2	91.35	99.12	90.54	99.11	90.56	98.21
0.2–0.4	92.40	100.0	92.17	100.0	92.56	100.0
0.4–0.6	92.68	100.0	92.59	100.0	92.98	100.0
0.6–0.8	92.57	99.17	92.74	99.17	92.74	99.17
0.8–1.0	89.44	100.0	92.60	100.0	92.15	100.0
1.0–1.2	86.57	93.52	91.63	100.0	90.87	100.0
1.2–1.4	83.28	81.15	89.82	98.36	89.62	97.54

the region of the conveyor visible to the robot's Kinect camera is only 1.4 m in length. Based on this length and for different speeds of the conveyor, we can easily compute the number of pose estimates each method would be able to obtain in the visible region (as shown in Table 7). It is evident from this table, that PERCH, due to its high runtime, is able to provide estimates for very low conveyor speeds only. Such speeds are not practical in a real-world industrial scenario where a high throughput is required. Moreover, if planning and execution time for the robot are taken into account, the PERCH pose estimates may be infeasible even for low speeds. In contrast, PERCH 2.0 can offer several pose estimates for the moving object even at high conveyor speeds due to its enhanced runtime. As a result, it can be used by planning methods such as Islam et al. (2021), which can utilize multiple pose estimates while planning for the arm.

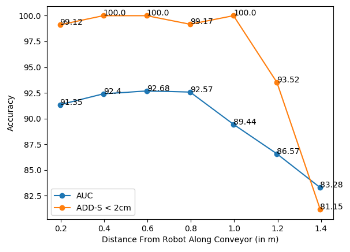
Spatial analysis. A key component of any method being applied to the given task is the variation in the method's accuracy as the object moves from end of the conveyor to the other end closer to the robot. A method that can offer higher accuracy even if the object is at the far end of the conveyor allows sufficient time for planning and execution of the pickup motion by the robot arm before the object reaches within the robot workspace. In order to understand the spatial variation in accuracy for each of the methods, we divide the conveyor into bins of 0.2 m and compute AUC as well as ADD-S $< 2\,\mathrm{cm}$ for poses belonging to each bin. The results are shown in Table 8 and Figure 18 for PERCH, PERCH 2.0 and BF-ICP. Figure 19 shows qualitative performance of PERCH 2.0 for different objects at varying distances from the robot. The overall observation consistent with all three methods is that the pose estimation accuracy reduces when the object is too close or too far from the robot. A close inspection of scenes when the object is far away revealed significant depth holes in the captured depth images which in-turn reduces the number of points available for pose estimation in the observed point-cloud (Figure 22a). Similarly for some objects in certain poses such as the 035_power_drill, the visible parts of the object when its close to the robot are insufficient to discern the correct 3-Dof pose.



(a) PERCH

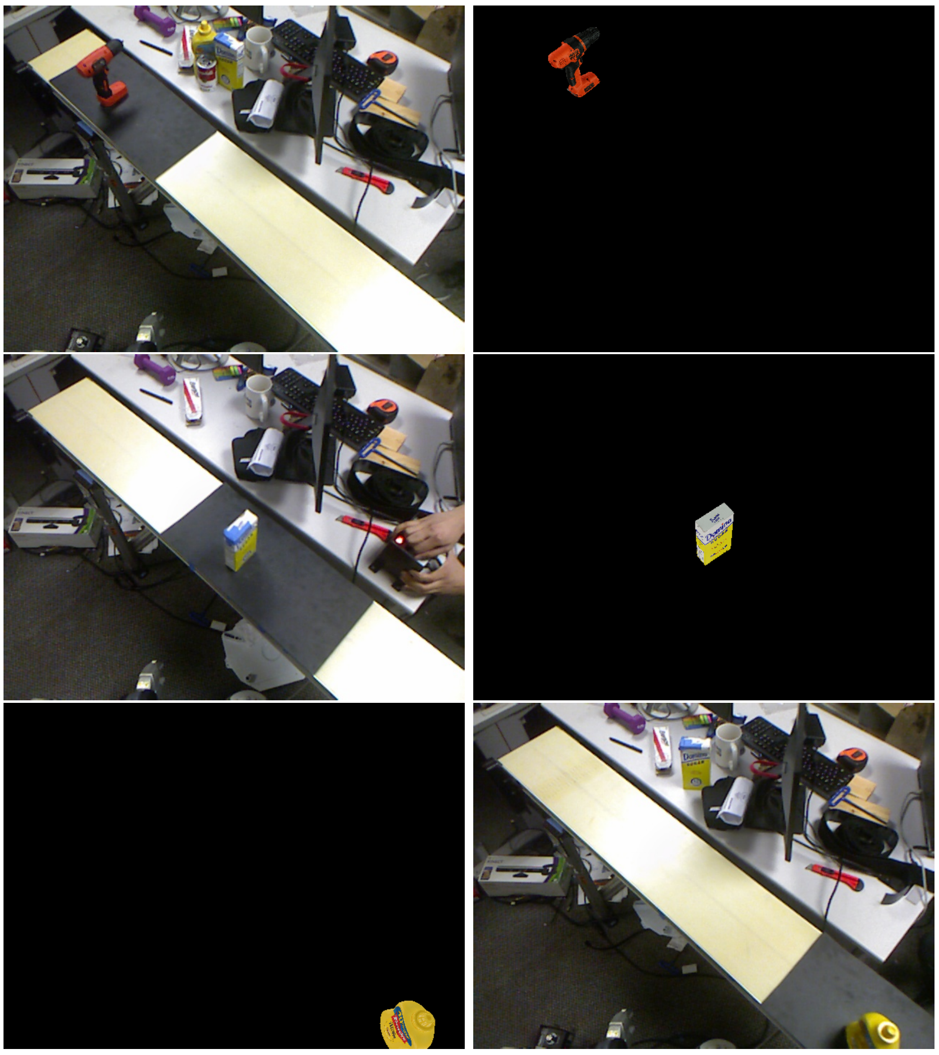


(b) PERCH 2.0



(c) BF-ICP

Figure 18. Variation of pose estimation accuracy with distance from robot along the conveyor.



(a) Input scene **(b)** Reconstructed scene from predicted poses

Figure 19. A few results from the conveyor dataset.

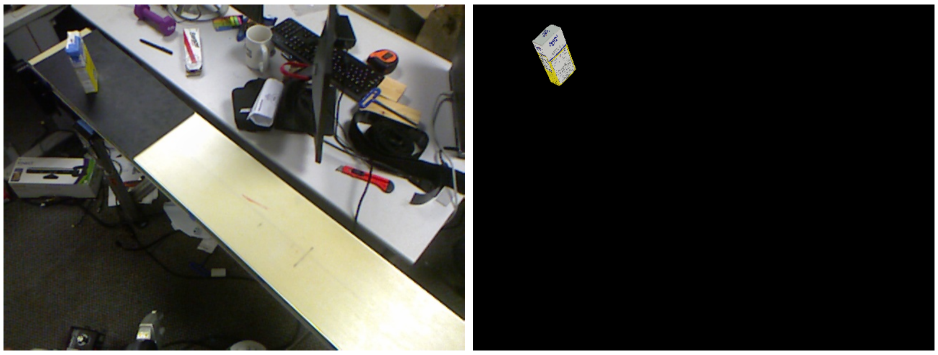

(a) Input scene (b) Reconstructed scene from predicted poses

Figure 20. PERCH 2.0 edge case from the Conveyor Dataset: Inaccurate pose for 004_sugar_box.

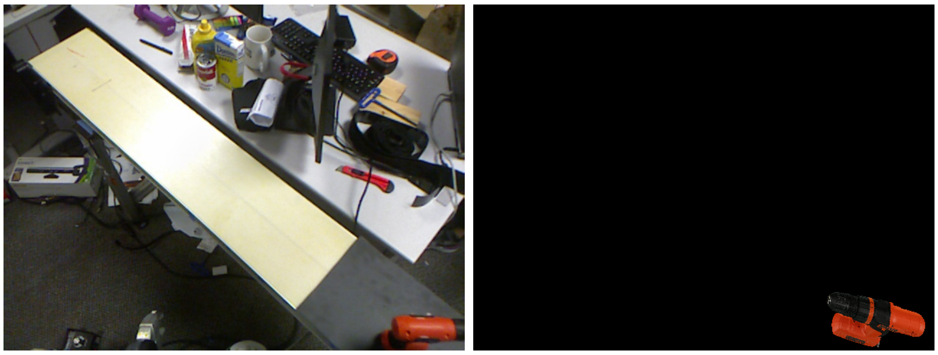
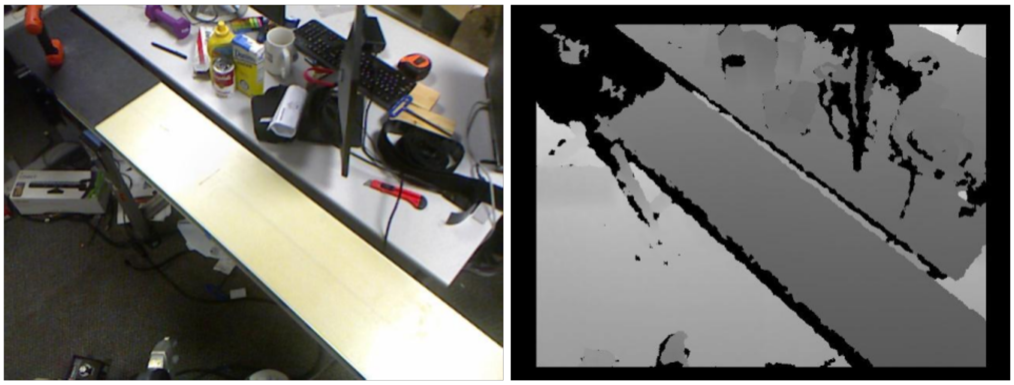

(a) Input scene (b) Reconstructed scene from predicted poses

Figure 21. PERCH 2.0 edge case from the Conveyor Dataset: Inaccurate pose for 035_power_drill.

This can be attributed to the creation of ambiguous top-down views of the object, again reducing the number of points available in the observed point-cloud for pose estimation (Figure 22b). Figures 20 and 21 represents the two edge cases where PERCH 2.0 predicts inaccurate or ambiguous poses. However, the second observation that is evident from Figure 18 is that the drop in accuracy when the object is far way is significantly higher for BF-ICP than for PERCH and PERCH 2.0. The drop clearly highlights that rendering plays a key role in determining the correct 3-Dof pose when objects are far from the camera and only few points from the object are actually visible. It also points to PERCH 2.0 being a favorable candidate for the given task owing to its low runtime when compared to PERCH and higher spatially consistent accuracy when compared to BF-ICP.

5.2.3. Container opening

Task. PERCH 2.0 was evaluated for the task of opening a container using the RoMan platform shown in Figure 25a. Here, PERCH 2.0 estimates the 3-Dof pose of the container (x, y, yaw) to be opened, which then serves as an input for the manipulation planner. The task of opening a container imposes stringent accuracy requirements on estimated pose due to manipulation work space limitations of the robot and the necessity to locate the handle of the container which occupies a limited area with respect to the entire container. In addition, the size of the container and its close proximity to the robot, renders large parts of the container occluded from the camera, thus making the pose estimation accuracy requirement harder to meet. While experiments conducted in Kessens et al. (2020) showed high overall container opening success rates when PERCH was used for


(a) Depth holes in far away objects

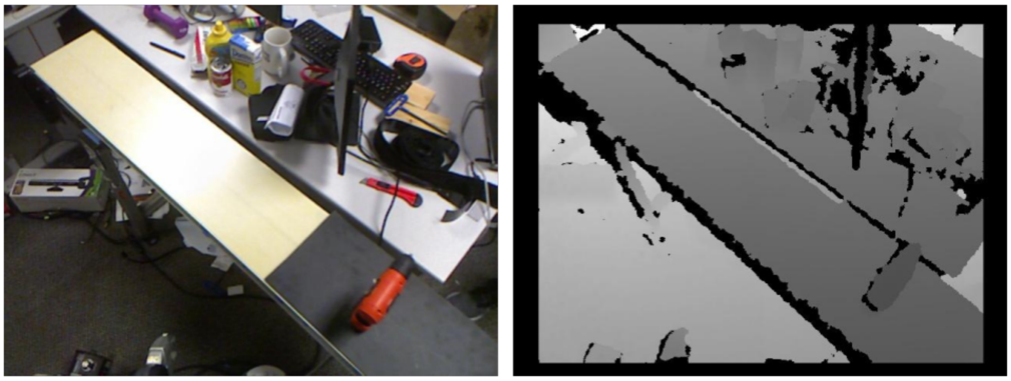

(b) Objects in close proximity result in ambiguous top-down views

Figure 22. Difficult scenarios for pose estimation of objects moving along the conveyor.

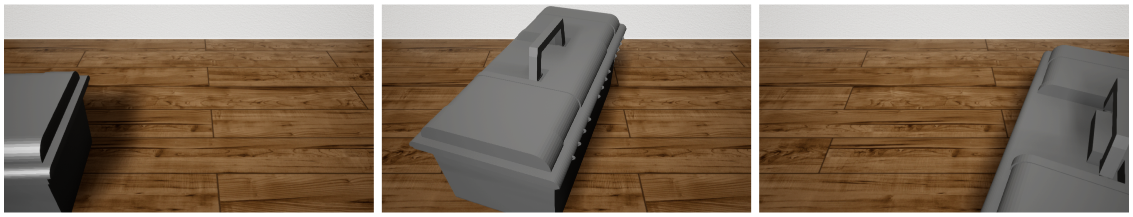


Figure 23. A few sample scenes from the container test dataset.

container pose estimation (over 80%), the high runtime still limits its applicability in the real world. This provides us the necessary motivation to consider PERCH 2.0 for this task.

Dataset. For evaluating PERCH 2.0 based pose estimation of the container and to compare its accuracy and runtime against the accuracy of PERCH, we require multiple scenes consisting of the container with varying (x, y, yaw) and different degrees of visibility. In addition, computation of accuracy, requires ground-truth pose-annotations for each scene, a task which is difficult to obtain in the real world. Subsequently, we constructed a synthetic photo-realistic dataset of 25 scenes containing the container with corresponding RGB and depth images using NVidia NDDS (To et al., 2018) plugin for Unreal Engine 4 (as shown in Figure 23) as before. Within the plugin, we randomly vary 3-Dof pose (x, y, yaw) of the container located on the ground. The height of the camera is fixed according to the height of the RealSense mounted on RoMan and scenes bearing close resemblance to real world requirements where the container is highly occluded and in close proximity to the RealSense are generated.

Baselines. In this experiment, we use two variants of PERCH 2.0:

- The first variant is PERCH 2.0 A (ICP + kNN I)
- The second variant is PERCH 2.0 B (M2M GICP + kNN II)

We compare these two variants with PERCH and BF-ICP baseline. For running PERCH 2.0, a machine with a P100 16GB GPU with 8 CPU cores is used. For PERCH a machine with 8 CPU cores is used.

Metrics. We use 3 metrics to compare our results—area under ADD-S threshold curve for every object ADD-S AUC < 0.1 m, the percentage of poses with ADD-S < 2 cm and the mean ADD-S error. The results are shown in Table 9.

Accuracy. The first observation is that PERCH 2.0 A and PERCH 2.0 B achieve comparable performance on the ADD-S < 2 cm metric with 96% of the poses within the 2 cm ADD-S error limit. However PERCH 2.0 B (M2M GICP + kNN II) performs significantly better than the other approaches on the metrics ADD-S AUC (< 0.1 m) and mean ADD-S error. This shows that GICP improves accuracy by performing better alignment under occlusion and is consistent with our observation in the experiment in Section 5.1. We note that all PERCH variants have a significantly better accuracy than the BF-ICP approach which highlights the importance of rendering and using the PERCH cost function in the estimation of the container pose especially when large parts of the container are occluded.

Runtime. The results show that PERCH 2.0 (M2M GICP + kNN II) is ∼50X faster that PERCH on the given dataset. It is also ∼1.5X faster than PERCH 2.0 (ICP + kNN Approach I) which shows that for a large number of candidate poses, parallel GICP as well as kNN Approach II help in reducing runtime further. This observation is also consistent with our observation in Section 5.1. For all methods the average number of poses rendered is within 700 to 800.

Qualitative results. Figure 24 shows a few scenes from crate database with corresponding predicted poses by PERCH 2.0. The predictions highlight the capability of PERCH 2.0 in being able to accurately estimate the 3-Dof pose even under high occlusion. Figure 25 represents scenes and predictions captured from the RoMan robot and show that the runtime and accuracy capabilities of PERCH 2.0 transfer well to the real world.

Table 9. Evaluation of 3-Dof pose estimation of container.

Objects	**BF-ICP**		**PERCH**		**PERCH 2.0 A (ICP + kNN I)**		**PERCH 2.0 B (M2M GICP + kNN II)**	
	AUC	**<2 cm**	**AUC**	**<2 cm**	**AUC**	**<2 cm**	**AUC**	**<2 cm**
container	51.12	24.00	88.41	96.00	90.35	96.00	**93.14**	**96.00**
Mean Runtime (s)	2.47		162.36		5.07		**3.23**	

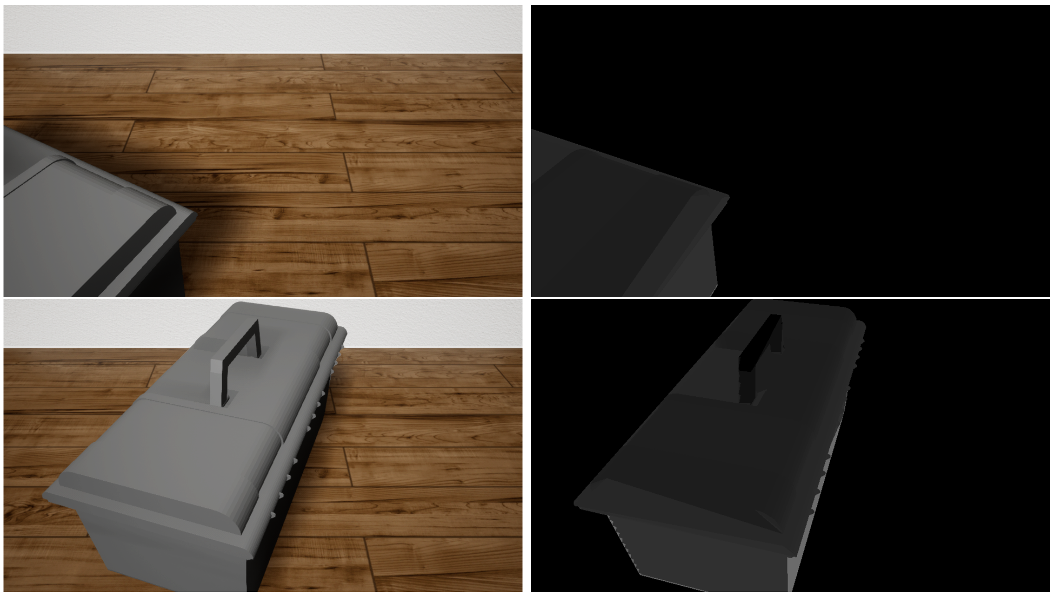
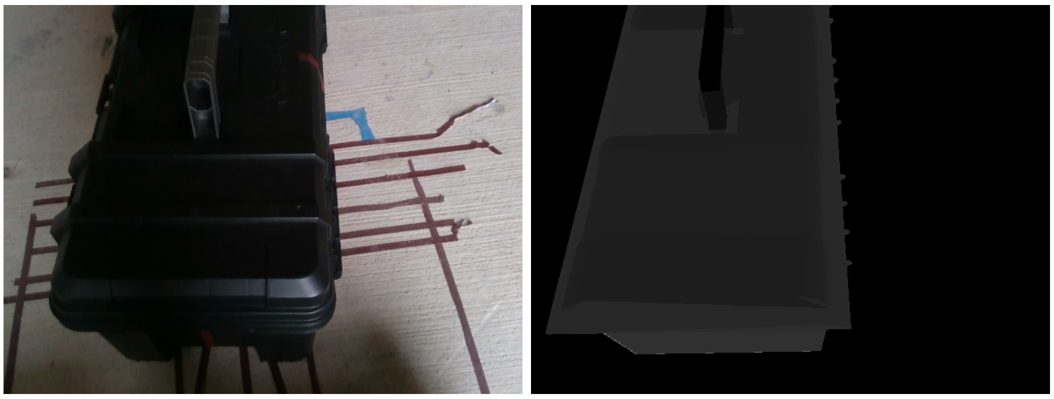


Figure 24. A few results from the crate test dataset.


(a) Input scene **(b)** Reconstructed scene (Runtime: 3.0 s)

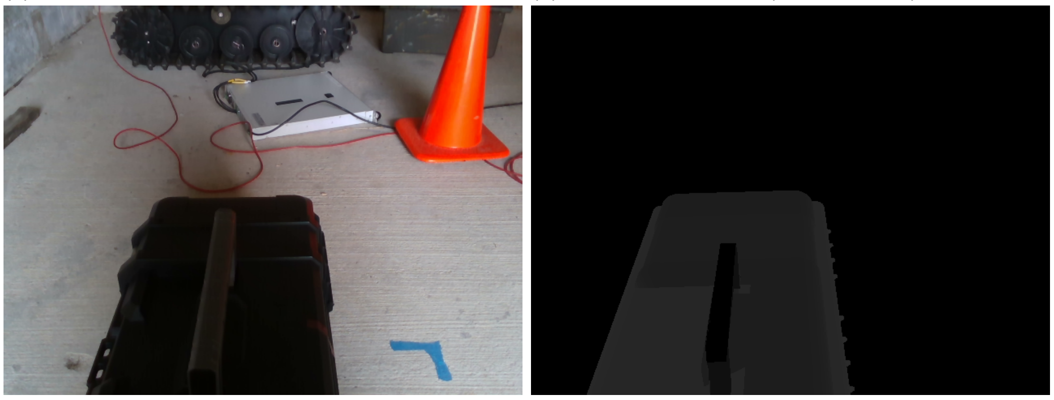

(c) Input scene **(d)** Reconstructed scene (Runtime: 1.4 s)

Figure 25. A few test scenes from the RoMan robot platform.

6. Conclusion

In this work we introduced PERCH 2.0, a parallel, GPU-based, deliberative, pose estimation methodology that finds the best explanation of an observed scene by searching a space of possible rendered scenes. The methodology scales proportionally with the number of objects and number of poses to be considered for each object by breaking down every task into the smallest possible unit that can fully utilize a GPU's parallelization capabilities.

Within the deliberative framework, we first render a set of candidate poses for every object of interest in the scene. Our GPU-based renderer can generate thousands of scenes at a time and inherently accounts for occlusion from other objects by using the input depth data to mark certain points in a given scene as extraneous occluders. Rendering is followed by 3D point-cloud creation wherein the camera's intrinsic parameters are used to simultaneously project every valid pixel in the rendered scenes to 3D space. The point-clouds are then fed into our many-to-many GICP framework that aligns every pose to the corresponding observed point-cloud. In order to counter the sequential nature of ICP iteration, we perform ICP iterations for all point-clouds in parallel, thereby reducing a multi-sequential step of aligning several point-clouds to a single sequential step. After GICP, the updated poses are rendered again and converted to point-clouds. In the final step, we evaluate the candidate poses by computing a cost function that relies on a parallel nearest neighbor search of every rendered 3D point in the observed scene.

Our experiments showed that PERCH 2.0 is two orders of magnitude faster than its predecessor PERCH (Narayanan & Likhachev, 2016b) for 3-Dof pose estimation. As a direct result of the speedup, we demonstrated the application of PERCH 2.0 to new domains such as container-opening and pose estimation of moving objects on a conveyor belt. For container opening, where the designated task was to estimate the 3-Dof pose of a large container, we showed that despite high occlusion due to close proximity of the container to the robot, our method is able to achieve high accuracy and low runtime. In the case of moving objects on a conveyor, PERCH 2.0 retains high accuracy even when the object is at the far end of the conveyor belt and offers several pose estimates as the object gets closer. Intuitively, both capabilities are favorable for a planner intending to grasp the object as it arrives within the robot workspace.

For 6-Dof pose estimation, where the number of poses to consider for every object increases exponentially, we demonstrated a successful and scalable integration of a discriminative learning based 2D object detector with the deliberative methodology PERCH 2.0. The combined framework predicts 6-Dof poses directly from an instance segmentation mask, generated by the 2D object detector, thus eliminating the need for ground-truth 6-Dof poses in the detector's training database. Our 6-Dof experiments demonstrated that our framework can achieve higher accuracy than state-of-the-art discriminative methods that do not have a deliberative component. Additionally, we showed the impact of localizing full bounding boxes, as opposed to visible ones, on overall pose estimation accuracy by better handling occlusion.

7. Future work

Our framework provides several directions for future work on deliberative pose estimation by formulating it in a more scalable and practically feasible manner.

For 3-Dof pose estimation, so far only indoor scenes have been explored in our work. This could be extended to outdoor scenes where Lidar data can be used to accurately estimate 3-Dof poses of vehicles by autonomous cars. Since outdoor scenes typically deal with higher distance magnitudes, a voxelized version of PERCH 2.0 may be developed to speed up computation even further. An extension to vehicle pose detection also presents an opportunity to extend deliberative pose estimation to scenarios where it may not be possible to have exact models of all objects.

In the context of 6-Dof pose estimation, we presented a framework that elegantly combines discriminative, deliberative, and optimization based approaches for pose prediction. However, in our method, predictions flow from the discriminative side to the deliberative side, but there is no feedback of information. As an extension, the costs predicted for each of the poses in the pose hypothesis could be used to learn a sampling distribution that takes the depth image of the scene as input. This sampling distribution can then be used to sample poses by PERCH 2.0 on the deliberative side instead of uniformly sampling the rotation space. This would enable a further speedup of the deliberative part of the framework and also ensure a continually improving self-supervised discriminative methodology that enables the deliberative part to get faster as it receives more data.

Acknowledgments

The authors thank Matt Kaplan and Trevor Rocks of the US Army Research Laboratory for their support. This work was conducted in part through collaborative participation in the Robotics Consortium sponsored by the U.S Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement W911NF-10-2-0016. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory of the U.S. Government.

ORCID

Maxim Likhachev https://orcid.org/0000-0002-9539-2398
Chad C. Kessens https://orcid.org/0000-0001-6366-0880

References

Agarwal, A., Han, Y., & Likhachev, M. (2020). PERCH 2.0: Fast and accurate GPU-based perception via search for object pose estimation. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 10633–10640. https://doi.org/10.1109/iros45743.2020.9341257

Aldoma, A., Tombari, F., Di Stefano, L., & Vincze, M. (2012). A global hypotheses verification method for 3D object recognition. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, & C. Schmid (Eds.), *Lecture Notes in Computer Science: Vol. 7574. Computer Vision – ECCV 2012* (pp. 511–524). Springer. https://doi.org/10.1007/978-3-642-33712-3_37

Aldoma, A., Tombari, F., Rusu, R. B., & Vincze, M. (2012). OUR-CVFH – oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6DOF pose estimation. In A. Pinz, T. Pock, H. Bischof, & F. Leberl (Eds.), *Lecture Notes in Computer Science: Vol. 7476. Pattern recognition. DAGM/OAGM 2012* (pp. 113–122). Springer. https://doi.org/10.1007/978-3-642-32717-9_12

Aldoma, A., Vincze, M., Blodow, N., Gossow, D., Gedikli, S., Rusu, R. B., & Bradski, G. (2011). CAD-model recognition and 6DOF pose estimation using 3D cues. *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 585–592. https://doi.org/10.1109/iccvw.2011.6130296

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517. https://doi.org/10.1145/361002.361007

Billings, G., & Johnson-Roberson, M. (2019). SilhoNet: An RGB method for 6D object pose estimation. *IEEE Robotics and Automation Letters*, *4*(4), 3727–3734. https://doi.org/10.1109/lra.2019.2928776

Buhrmester, M., Kwang, T., & Gosling, S. D. (2011). Amazon's Mechanical Turk: A new source of inexpensive, yet high-quality, data? *Perspectives on Psychological Science*, *6*(1), 3–5. https://doi.org/10.1177/1745691610393980

Calli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., & Dollar, A. M. (2015). Benchmarking in manipulation research: Using the Yale-CMU-Berkeley object and model set. *IEEE Robotics and Automation Magazine*, *22*(3), 36–52. https://doi.org/10.1109/mra.2015.2448951

Cao, Z., Sheikh, Y., & Banerjee, N. K. (2016). Real-time scalable 6DOF pose estimation for textureless objects. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2441–2448. https://doi.org/10.1109/icra.2016.7487396

Chen, Y., & Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, *10*(3), 145–155. https://doi.org/10.1016/0262-8856(92)90066-c

Choi, C., & Christensen, H. I. (2016). RGB-D object pose estimation in unstructured environments. *Robotics and Autonomous Systems*, *75*, 595–613. https://doi.org/10.1016/j.robot.2015.09.020

Corona, E., Kundu, K., & Fidler, S. (2018). Pose estimation for objects with rotational symmetry. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 7215–7222. https://doi.org/10.1109/iros.2018.8594282

Cowley, A., Cohen, B., Marshall, W., Taylor, C. J., & Likhachev, M. (2013). Perception and motion planning for pick-and-place of dynamic objects. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 816–823. https://doi.org/10.1109/iros.2013.6696445

Deng, X., Mousavian, A., Xiang, Y., Xia, F., Bretl, T., & Fox, D. (2021). PoseRBPF: A Rao-Blackwellized particle filter for 6D object pose tracking. *IEEE Transactions on Robotics*, *37*(5), 1328–1342. https://doi.org/10.1109/tro.2021.3056043

Deng, X., Xiang, Y., Mousavian, A., Eppner, C., Bretl, T., & Fox, D. (2020). Self-supervised 6D object pose estimation for robot manipulation. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 3665–3671. https://doi.org/10.1109/icra40945.2020.9196714

Garcia, V., Debreuve, É., Nielsen, F., & Barlaud, M. (2010). K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. *2010 IEEE International Conference on Image Processing (ICIP)*, 3757–3760. https://doi.org/10.1109/icip.2010.5654017

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 580–587. https://doi.org/10.1109/cvpr.2014.81

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2980–2988. https://doi.org/10.1109/ICCV.2017.322

Hinterstoisser, S., Cagniart, C., Ilic, S., Sturm, P., Navab, N., Fua, P., & Lepetit, V. (2012). Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(5), 876–888. https://doi.org/10.1109/tpami.2011.206

Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., & Navab, N. (2013). Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In K. M. Lee, Y. Matsushita, J. M. Rehg, & Z. Hu (Eds.), *Lecture Notes in Computer Science: Vol. 7724. Computer Vision – ACCV 2012* (pp. 548–562). Springer. https://doi.org/10.1007/978-3-642-37331-2_42

Islam, F., Salzman, O., Agarwal, A., & Likhachev, M. (2021). Provably constant-time planning and replanning for real-time grasping objects off a conveyor belt. *The International Journal of Robotics Research*. Advance online publication. https://doi.org/10.1177/02783649211027194

Johnson, A. E., & Hebert, M. (1999). Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21*, 433–449. https://doi.org/10.1109/34.765655

Kehl, W., Manhardt, F., Tombari, F., Ilic, S., & Navab, N. (2017). SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again. *2017 IEEE International Conference on Computer Vision (ICCV)*, 1530–1538. https://doi.org/10.1109/iccv.2017.169

Kessens, C. C., Fink, J., Hurwitz, A., Kaplan, M., Osteen, P. R., Rocks, T., Rogers, J., Stump, E., Quang, L., DiBlasi, M., Gonzalez, M., Patel, D., Patel, J., Patel, S., Weiker, M., Bowkett, J., Detry, R., Karumanchi, S., Burdick, J., . . . Srinivasa, S. (2020). Toward fieldable human-scale mobile manipulation using RoMan. In T. Pham, L. Solomon, & K. Rainey (Eds.), *Proceedings SPIE 11413, Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II* (pp. 418–437). SPIE. https://doi.org/10.1117/12.2559995

Koide, K., Yokozuka, M., Oishi, S., & Banno, A. (2021). Voxelized GICP for fast and accurate 3D point cloud registration. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 11054–11059. https://doi.org/10.1109/ICRA48506.2021.9560835

Labbé, Y., Carpentier, J., Aubry, M., & Sivic, J. (2020). CosyPose: Consistent multi-view multi-object 6D pose estimation. In A. Vedaldi, H. Bischof, T. Brox, & J.-M. Frahm (Eds.), *Lecture Notes in Computer Science: Vol. 12362. Computer Vision – ECCV 2020* (pp. 574–591). https://doi.org/10.1007/978-3-030-58520-4_34

Liu, J., & He, S. (2019). *6D object pose estimation without PnP*. arXiv: 1902.01728 [cs.CV].

Lowe, D. G. (1999). Object recognition from local scale-invariant features. *1999 IEEE International Conference on Computer Vision (ICCV)*, *2*, 1150–1157. https://doi.org/10.1109/iccv.1999.790410

Marion, P., Florence, P. R., Manuelli, L., & Tedrake, R. (2018). Label fusion: A pipeline for generating ground truth labels for real RGBD data of cluttered scenes. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 3235–3242. https://doi.org/10.1109/icra.2018.8460950

Massa, F., & Girshick, R. (2018). *maskrcnn-benchmark: Fast, modular reference implementation of instance segmentation and object detection algorithms in PyTorch* [Computer software]. GitHub. https://github.com/facebookresearch/maskrcnn-benchmark

Menon, A., Cohen, B., & Likhachev, M. (2014). Motion planning for smooth pickup of moving objects. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 453–460. https://doi.org/10.1109/icra.2014.6906895

Mitash, C., Bekris, K. E., & Boularias, A. (2017). A self-supervised learning system for object detection using physics simulation and multi-view pose estimation. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 545–551. https://doi.org/10.1109/iros.2017.8202206

Mitash, C., Boularias, A., & Bekris, K. E. (2018). Improving 6D pose estimation of objects in clutter via physics-aware Monte Carlo tree search. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 3331–3338. https://doi.org/10.1109/icra.2018.8461163

Mörwald, T., Prankl, J., Richtsfeld, A., Zillich, M., & Vincze, M. (2010). BLORT – the blocks world robotic vision toolbox. *Proceedings ICRA Workshop Best Practice in 3D Perception and Modeling for Mobile Manipulation.*

Mousavian, A., Anguelov, D., Flynn, J., & Kosecka, J. (2017). 3D bounding box estimation using deep learning and geometry. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5632–5640. https://doi.org/10.1109/cvpr.2017.597

Narayanan, V., & Likhachev, M. (2016a). Discriminatively-guided deliberative perception for pose estimation of multiple 3D object instances. *Proceedings of Robotics: Science and Systems (RSS '16).*

Narayanan, V., & Likhachev, M. (2016b). PERCH: Perception via search for multi-object recognition and localization. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, *2016-June*, 5052–5059. https://doi.org/10.1109/icra.2016.7487711

Narayanan, V., & Likhachev, M. (2017). Deliberative object pose estimation in clutter. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 3125–3130. https://doi.org/10.1109/ICRA.2017.7989357

Pavlakos, G., Zhou, X., Chan, A., Derpanis, K. G., & Daniilidis, K. (2017). 6-DOF object pose from semantic keypoints. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2011–2018. https://doi.org/10.1109/icra.2017.7989233

Qiu, D., May, S., & Nüchter, A. (2009). GPU-accelerated nearest neighbor search for 3D registration. In M. Fritz, B. Schiele, & J. H. Piater (Eds.), *Lecture Notes in Computer Science: Vol. 5815. Computer Vision Systems. ICVS 2009* (pp. 194–203). Springer. https://doi.org/10.1007/978-3-642-04667-4_20

Rad, M., & Lepetit, V. (2017). BB8: a scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth. *2017 IEEE International Conference on Computer Vision (ICCV)*, 3848–3856. https://doi.org/10.1109/iccv.2017.413

Rothganger, F., Lazebnik, S., Schmid, C., & Ponce, J. (2006). 3D object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *International Journal of Computer Vision*, *66*(3), 231–259. https://doi.org/10.1007/s11263-005-3674-1

Rusu, R. B., Blodow, N., & Beetz, M. (2009). Fast point feature histograms (FPFH) for 3D registration. *2009 IEEE International Conference on Robotics and Automation (ICRA)*, 3212–3217. https://doi.org/10.1109/robot.2009.5152473

Rusu, R. B., Bradski, G., Thibaux, R., & Hsu, J. (2010). Fast 3D recognition and pose using the viewpoint feature histogram. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2155–2162. https://doi.org/10.1109/iros.2010.5651280

Rusu, R. B., & Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 1–4. https://doi.org/10.1109/icra.2011.5980567

Saxena, D. M., Saleem, M. S., & Likhachev, M. (2021). Manipulation planning among movable obstacles using physics-based adaptive motion primitives, 6570–6576. https://doi.org/10.1109/icra48506.2021.9561221

Segal, A., Hähnel, D., & Thrun, S. (2009). Generalized-ICP. *Robotics: science and systems*, *2*(4), 435. https://doi.org/10.15607/rss.2009.v.021

Shao, S., Zhao, Z., Li, B., Xiao, T., Yu, G., Zhang, X., & Sun, J. (2018). *CrowdHuman: A benchmark for detecting human in a crowd.* arXiv: 1805.00123 [cs.CV].

Sharma, G., Wu, W., & Dalal, E. N. (2005). The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, *30*(1), 21–30. https://doi.org/10.1002/col.20070

Stevens, M. R., & Beveridge, J. R. (2000a). *The Springer International Series in Engineering and Computer Science: Vol. 589. Integrating graphics and vision for object recognition.* Springer. https://doi.org/10.1007/978-1-4757-5524-4

Stevens, M. R., & Beveridge, J. R. (2000b). Localized scene interpretation from 3D models, range, and optical data. *Computer Vision and Image Understanding*, *80*(2), 111–129. https://doi.org/10.1006/cviu.2000.0821

Sundermeyer, M., Marton, Z.-C., Durner, M., Brucker, M., & Triebel, R. (2018). Implicit 3D orientation learning for 6D object detection from RGB images. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Lecture Notes in Computer Science: Vol. 11210. Computer Vision – ECCV 2018* (pp. 699–715). Springer.

Suwajanakorn, S., Snavely, N., Tompson, J. J., & Norouzi, M. (2018). Discovery of latent 3D keypoints via end-to-end geometric reasoning. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2063–2074.

Tekin, B., Sinha, S. N., & Fua, P. (2018). Real-time seamless single shot 6D object pose prediction. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 292–301. https://doi.org/10.1109/cvpr.2018.00038

To, T., Tremblay, J., McKay, D., Yamaguchi, Y., Leung, K., Balanon, A., Cheng, J., Hodge, W., & Birchfield, S. (2018). *NDDS: NVIDIA deep learning dataset synthesizer* [Computer software]. GitHub. https://github.com/NVIDIA/Dataset_Synthesizer

Tombari, F., Salti, S., & Di Stefano, L. (2010). Unique signatures of histograms for local surface description. In K. Daniilidis, P. Maragos, & N. Paragios (Eds.), *Lecture Notes in Computer Science: Vol. 6313. Computer Vision – ECCV 2010* (pp. 356–369). Springer. https://doi.org/10.1007/978-3-642-15558-1_26

Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., & Birchfield, S. (2018). *Deep object pose estimation for semantic robotic grasping of household objects.* arXiv: 1809.10790 [cs.RO].

Tulsiani, S., & Malik, J. (2015). Viewpoints and keypoints. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1510–1519. https://doi.org/10.1109/cvpr.2015.7298758

Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., & Savarese, S. (2019). DenseFusion: 6D object pose estimation by iterative dense fusion. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, *2019-June*, 3338–3347. https://doi.org/10.1109/cvpr.2019.00346

Wohlhart, P., & Lepetit, V. (2015). Learning descriptors for object recognition and 3D pose estimation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3109–3118. https://doi.org/10.1109/cvpr.2015.7298930

Wong, J. M., Kee, V., Le, T., Wagner, S., Mariottini, G.-L., Schneider, A., Hamilton, L., Chipalkatty, R., Hebert, M., Johnson, D. M. S., Wu, J., Zhou, B., & Torralba, A. (2017). SegICP: Integrated deep semantic segmentation and pose estimation. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5784–5789. https://doi.org/10.1109/iros.2017.8206470

Xiang, Y., Schmidt, T., Narayanan, V., & Fox, D. (2018, June). PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. In H. Kress-Gazit, S. S. Srinivasa, T. Howard, & N. Atanasov (Eds.), *Proceedings of Robotics: Science and Systems XIV*. https://doi.org/10.15607/rss.2018.xiv.019

How to cite this article: Agarwal, A., Oza, Y., Likhachev, M., & Kessens, C. C. (2021). Fast and high-quality, GPU-based, deliberative, object-pose estimation. *Field Robotics*, *1*, 34–69.