A Correct-and-Certify Approach to Self-Supervise Object Pose Estimators via Ensemble Self-Training

Jingnan Shi*, Rajat Talak*, Dominic Maggio, and Luca Carlone

Abstract—Real-world robotics applications demand object pose estimation methods that work reliably across a variety of scenarios. Modern learning-based approaches require large labeled datasets and tend to perform poorly outside the training domain. Our first contribution is to develop a *robust corrector* module that corrects pose estimates using depth information, thus enabling existing methods to better generalize to new test domains; the corrector operates on semantic keypoints (but is also applicable to other pose estimators) and is fully differentiable. Our second contribution is an *ensemble self-training* approach that simultaneously trains multiple pose estimators in a self-supervised manner. Our ensemble self-training architecture uses the robust corrector to refine the output of each pose estimator; then, it evaluates the quality of the outputs using *observable correctness* certificates; finally, it uses the observably correct outputs for further training, without requiring external supervision. As an additional contribution, we propose small improvements to a regression-based keypoint detection architecture, to enhance its robustness to outliers; these improvements include a robust pooling scheme and a robust centroid computation. Experiments on the YCBV and TLESS datasets show the proposed ensemble self-training performs on par or better than fully supervised baselines while not requiring 3D annotations on real data. Code and pre-trained models are available on the project web page[1].

I. INTRODUCTION

Object pose estimation is a key capability for robotics. From search-and-rescue in caves and subways [1], to domestic robotics and factory automation [2], to satellite pose estimation for autonomous docking and debris removal [3], estimating 3D poses of external objects is a necessary prerequisite for robot navigation, manipulation, and human-robot interaction.

Recent progress in learning-based object pose estimation has been fueled by the availability of pose-annotated datasets [4], [5] and computer vision challenges (*e.g.,* the BOP challenge [6], [7]). However, in robotics applications it is not always practical to collect training data for all relevant environments, also considering that most techniques require *3D* pose annotations, which are more challenging to manually annotate. While increasingly more realistic simulators are reducing the sim-to-real gap, it remains unclear if the gap
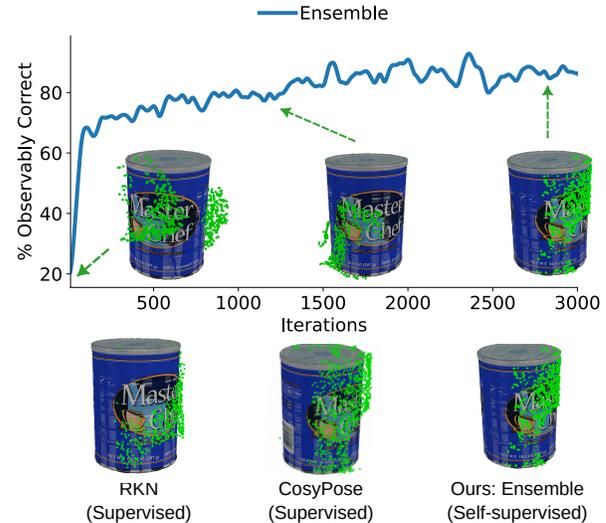
Fig. 1: We propose an *ensemble self-training* architecture that simultaneously trains multiple pose estimators without external supervision. (*Top*) The percentage of observably correct pose estimates increases as the self-training progresses, jumping from 19% to 80% after 3000 iterations. (*Bottom*) The resulting estimates are on par or better compared to fully supervised baselines, including a point-cloud based architecture (RKN) and CosyPose [8]. Green points represent the input point cloud transformed into the model frame based on the estimated pose.

will eventually disappear, and while simulation makes it easy to provide ground-truth labels for the object poses, creating a variety of assets to cover different scenarios remains an expensive and time-consuming task. Therefore, it would be desirable to develop self-supervised learning approaches that can learn from real data without manual 3D annotations.

A related issue is that current learning-based techniques generalize poorly outside the training domain. For instance, a change in the placement of the camera (*e.g.,* from a ground robot to a drone) may induce a domain gap. Similarly, an approach trained in simulation might exhibit a sim-to-real gap, hindering real-world performance. Therefore, it would be desirable to design approaches that can better bridge the gap between training and testing, and generalize to unseen data.

In this paper we tackle these issues with two main contributions. First, we propose a **robust corrector** (Section IV), which can be added to the output of any pose estimation architecture (as long as a CAD model of the object is available) and is able to correct moderate pose estimation errors. The robust corrector is a differentiable optimization layer, and corrects the pose output by using depth or point-cloud data and

the given object CAD model. While Talak *et al.* [9] showed that a similar scheme outperforms standard pose refinement methods (*e.g.,* ICP), the key feature of our method is its robustness to outliers, *i.e.,* points incorrectly segmented to be part of the object. Additionally, the robust corrector enables a simple differentiation rule for back-propagation.

Our second contribution is an **ensemble self-training** approach (Section V), in which, any number of existing object pose estimation models can be augmented and self-trained, in parallel, on real-world data without requiring pose annotations. The ensemble self-training approach attaches a robust corrector to each pose estimator in order to refine their outputs. Then, it computes *certificates of observable correctness* to assess the quality of the output of each pose estimator. An observable correctness certificate is a binary condition that checks whether a pose estimate (after being refined by the corrector) meets some basic geometric consistency requirements. We impose two such consistency checks. The first, requires that the posed CAD model (posed at the corrector pose estimate) matches the input. The second, requires that a rendering of the posed CAD model matches a 2D segmentation mask of the object. These certificates extend the corresponding observable correctness certificates in [9] to account for the presence of outliers and to operate on both 3D and 2D data. The corrector pose, output by the robust corrector, is said to be observably correct, if it meets both these geometric consistency checks. Finally, the ensemble self-training approach selects the observably correct outputs across all the pose estimators in the ensemble, and uses them for further self-supervised training. The key idea is that when an estimator produces an observably correct output, it can learn from it, while simultaneously informing the other estimators. This allows exploiting the complementarity between different models (*e.g.,* a point-cloud-based architecture might perform better than an RGB-based one in low-lighting conditions), while helping to bootstrap the self-training process (*e.g.,* if a model initially is not able to get observably correct instances, it still gets informed by observably correct instances produced by the other models). In our experiments, we observe that as the self-training progresses the number of observably correct instances increases, and the proposed ensemble self-training approach outperforms each model in the ensemble, even when those are provided with full supervision (Fig. 1).

As an addition contribution, we develop an **outlier-robust point cloud processing** by proposing small improvements to a standard point-cloud-based regression model for semantic-keypoint detection (Section VI). Regression models for semantic-keypoint detection (*e.g.,* [9]) are not robust to outliers in the segmentation mask due to two reasons: (i) most of the point cloud architectures use farthest point sampling (FPS) [10], [11], [12], which in the presence of outliers ends up sampling many outlier points, rather than rejecting them; (ii) the lack of exact translation and rotation equivariance of point-cloud-based architectures results in incorrect detections, when the input point cloud is not centered correctly — a phenomenon that happens often when the point cloud contains many outliers. We propose a trainable **robust pooling** layer

and a **robust centroid** computation, that can make any point cloud-based model robust to outliers. The robust pooling samples points based on regressed scores, using point features and trainable weights. The robust centroid computation computes a robust center for the input point cloud using graduated non-convexity [13]. The point cloud is then centered at this robust center, before passing it to the point cloud-based model.

We conclude the paper by evaluating the robust corrector, the ensemble self-training, and the outlier-robust point cloud processing on the YCBV [14], [7] and the TLESS [5], [7] datasets. The results confirm that (i) the robust corrector is able to correct large errors in the keypoint detections; (ii) the ensemble self-training produces pose estimates on par or better compared to fully supervised baselines (10% average increase in the ADD-S (AUC) scores over CosyPose [8] on selected YCBV objects), and largely outperforms competing self-supervised methods (40% increase in the ADD-S (AUC) scores over Self6D++ [15]); (iii) the outlier-robust point cloud processing increases the robustness of the keypoint detection in the presence of outliers. The code will be released upon the acceptance of this paper.

Before delving into our contributions, we review related work (Section II) and formally state the problem (Section III).

II. RELATED WORKS

Self-Supervised Object Pose Estimation. The literature on self-supervised pose estimation is sparse and very recent. Wang *et al.* [16] train a pose estimation model on synthetic RGB-D data, and then refine it further with self-supervised training on real, unannotated data; differentiable rendering provides the required supervision signal. Self6D++ [15] extends [16] by accounting for possible occlusions of the object. Chen *et al.* [17] propose a student-teacher iterative scheme to bridge the sim-to-real domain gap, for a sim-trained pose estimation model. Wang *et al.* [18] extract pose-invariant features, thereby canonizing object's shape, and use them for category-level object pose estimation of unseen object instances. Zakharov *et al.* [19] utilize differentiable rendering of signed distance fields of objects, along with normalized object coordinate spaces [18], to learn 9D cuboids in a self-supervised manner. Zhang *et al.* [20] propose to jointly reconstruct the 3D shape of an object category and learn dense 2D-3D correspondences between the input image and the 3D shape, by training on large-scale, real-world object videos. Deng *et al.* [21] self-supervise pose estimation by interacting with the objects in the environment; the model gets trained on the data collected autonomously by a manipulator.

Point-cloud-only, self-supervised approaches for pose estimation have been also proposed. Li *et al.* [22] extract an $SE(3)$-invariant feature, which works as a canonical object, and use it to supervise training with a Chamfer loss. Sun *et al.* [23] tackle self-supervised point cloud alignment by extracting features using capsule network. Talak *et al.* [9] propose a self-supervised keypoint-based pose estimator that uses a corrector and binary certificates during self-supervision.

Our approach shares insights with [17], [16], [15], [9]. Similar to these approaches, our goal is to take sim-trained models and further train them in a self-supervised manner. The works [15], [17] use a student-teacher architecture which requires instantiating two networks: the teacher and the student; on the other hand, with the proposed corrector and certificates we can directly self-supervise the pose estimation models without the need to instantiate another network. The works [9], [17] use certificates or consistency checks to assess the quality of the pose estimates for self-training; contrary to our certificates, the ones in [9], [17] are sensitive to outliers resulting from a noisy segmentation of the object, due to their reliance on non-robust distance functions. Conversely, [16] is not robust to foreground occlusions of the object, as shown in [15]. The work [9] is the first to propose a self-training procedure based on a corrector and binary certificates. However, the approach is restricted to keypoint-based methods and performs poorly in the presence of outliers (Section VII). In addition, none of the works above jointly self-supervise multiple models.

Point Cloud Architectures. The success of convolutional neural networks on images has led many researchers to investigate models that can work directly on point cloud data (*e.g.,* produced by a LiDAR or an RGB-D camera). However, most existing point-cloud-based models suffer from the fact that there is not an easy and efficient equivalent to the pooling/unpooling operation for point cloud data. Most models use farthest point sampling (FPS) (which sequentially samples a set of points, such that each point is farthest away from all the points sampled thus far). This method often leads to sub-optimal performance [24], [25], [26], [27], [28], [29], [30], [31], especially in the presence of outliers; see also our analysis in Section VII. Dovrat *et al.* [31] show that a task-specific, learning-based approach to sub-sample points can outperform a method like FPS. Yang *et al.* [30] propose a learnable, task-agnostic, Gumbel subset sampling, which produces "soft" subsets in training, and hard discrete subsets at test-time. Lang *et al.* [29] propose a differentiable relaxation of point cloud sampling, as a mixture of points. Nezhadarya *et al.* [28] propose a global down-sampling method that first determines critical points to sample, from regressed point features, hence obviating the need to use computationally expensive k-nearest neighbor-based aggregation. Yan *et al.* [27] propose adaptive sampling, that modifies the sampled points using FPS, and then uses a local-nonlocal module to extract and capture neighboring and long-range dependencies of the sampled points. Lin *et al.* [25] propose a learning-based sampling strategy, and shows how the sampled points differ across various tasks such as object-part segmentation and point cloud completion — making the point that the sampling strategy needs to be task specific. Wang *et al.* [24] propose a lightweight transformer network for point cloud downsampling. None of these works investigate the effect of outliers in point-cloud-based regression models for semantic keypoint detection.

Differentiable Optimization. A differentiable optimization layer solves an optimization problem, and also computes the gradient of the optimal solution with respect to the
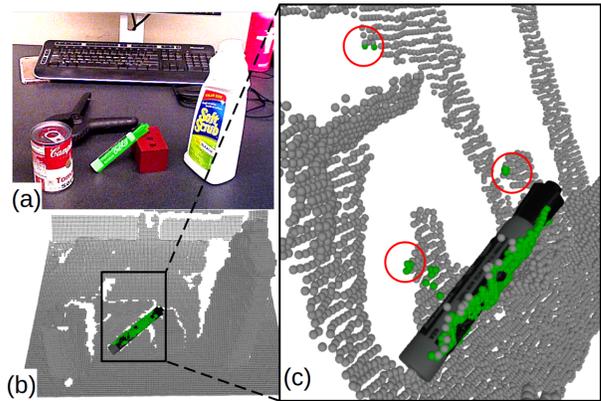

Fig. 2: Examples of outliers caused by a noisy 2D segmentation. (a) RGB image with segmentation mask overlaid in green. (b) Depth map with masked points highlighted in green. (c) Zoomed-in view of the depth point cloud, with outliers circled in red. Outliers are common even in manually annotated masks: the example in the figure is obtained using the ground-truth segmentation available in the YCBV dataset.

input parameters, for back-propagation. A differentiable optimization layer allows a learning-based model to explicitly take into account various geometric and physical constraints. Related work has developed tools to differentiate through quadratic optimization problems [32], convex problems [33], non-linear optimization problems [34], stochastic optimization problems [35], and combinatorial optimization problems [36], [37], [38]. Open-source libraries, such as [39], [40], provide off-the-shelf tools to implement differentiable non-linear least-squares. Our architecture implements a robust corrector module as a differentiable optimization layer. Although, it solves a non-convex optimization problem, we are able to exploit its specific structure to implement a very simple derivative.

III. PROBLEM STATEMENT

Consider a robot equipped with a calibrated RGB-D camera. The camera collects color and depth data picturing a 3D scene containing an object of interest; we assume the object has a known shape, *i.e.,* we have its CAD model. We also assume access to a standard pre-trained 2D object detection and instance segmentation model (*e.g.,* pre-trained MaskRCNN [41]), and use it to extract a region of interest $\boldsymbol{I}$ (RoI) and the object 2D segmentation mask $\boldsymbol{M}$. Given the 2D segmentation mask and camera intrinsics, we compute the partial point cloud $\boldsymbol{X}$ of the detected object, along with its (RGB) color $\boldsymbol{f}$ as:

$$\boldsymbol{X}, \boldsymbol{f} = \pi^{-1}\left(\boldsymbol{I}, \boldsymbol{M}\right), \tag{1}$$

where $\pi^{-1}\left(\cdot\right)$ simply back-projects the pixels in the mask $\boldsymbol{M}$ to 3D points using the depth and camera intrinsics, and attaches (RGB) color information to each point. Here, $\boldsymbol{X} \in \mathbb{R}^{3 \times n}$ and $\boldsymbol{f} \in \mathbb{R}^{3 \times n}$, where n is the number of points.

The segmentation mask $\boldsymbol{M}$—produced by the 2D segmentation model— is typically noisy and may include outlier pixels that do not actually belong to the object or miss pixels that

do belong to it (Fig. 2). More formally, if M is a segmented 2D binary mask, then we can write it as:

$$M = M^* + O_M, \qquad (2)$$

where M^* denotes the ground-truth mask for the object and O_M contains the misclassified (outlier) pixels.

The outliers in the segmentation mask, in turn, induce outliers in the partial point cloud X of the object, according to eq. (1); these are 3D points that do not actually belong to the object (Fig. 2). To formalize this concept, it is useful to define a generative model that relates the point cloud X with the to-be-computed object pose T^* and its CAD model B. We write the generative model for the partial point cloud X as follows:

$$X = \Theta\left(T^* \cdot B\right) + n_w + O_X, \qquad (3)$$

where $T^* \cdot B$ is the *ground-truth posed CAD model* (*i.e.,* the CAD model arranged according to its true pose T^*), $\Theta(\cdot)$ is an occlusion function that samples a subset of points on the posed CAD model $T^* \cdot B$ of the object, restricted to the ground-truth mask M^*, n_w is the sensor measurement noise, and O_X are the outlier points caused by outlier pixels in O_M.

Our goal is to develop an estimator for the object pose T^*. We consider the realistic case where we have access to existing learning-based pose estimation models, but these have been trained on a different domain (*e.g.,* they have been trained in simulation). Therefore, our goal is to further train these models on real data without 3D annotations and in the presence of a potentially large domain gap (*e.g.,* sim-to-real gap).

Problem 1 (Self-Train Pose Estimation Models)**.** *Propose a method to take one or more sim-trained pose estimation models, and train them, in parallel, with self-supervision on unannotated real-world data.*

IV. ROBUST CORRECTOR

In this section we focus on keypoint-based pose estimation and we develop a *robust corrector* that can compensate errors in the keypoint detections (*e.g.,* caused by the sim-to-real gap). In the next section we show that the same ideas can be applied to other pose estimators that do not rely on keypoints.

A semantic-keypoint-based pose estimator (*e.g.,* [42], [43], [44]) first uses a neural network to detect semantic keypoints $\tilde{y}$ from the input; the neural network is trained to detect specific points on the surface of the object (see Fig. 3). The detector can be implemented using CNN-based architectures on RGB inputs (*e.g.,* [43], [44]) or regression models (*e.g.,* point transformers [12]) on point cloud and RGB-D data. After detecting the keypoints, a pose estimate can be retrieved via point-cloud registration, *i.e.,* by computing the rigid transformation $\tilde{T}$ that aligns the annotated keypoints b on the CAD model (Fig. 3), and the

Fig. 3: CAD model with annotated semantic keypoints.

detected keypoints $\tilde{y}$. In this section we will assume the keypoint detector to be given, while in Sections V and VI we will discuss how to self-train a detector and how to improve the robustness of existing architectures, respectively.

Robust Corrector Overview. The detected keypoints $\tilde{y}$ may be inaccurate when there is a large domain gap, which in turn leads to poor pose estimates $\tilde{T}$. Our first contribution is a *robust corrector*, that corrects the detected keypoints by utilizing the dense information in the input point cloud X and the object CAD model B. Contrary to related work [9], the robust corrector is designed to automatically reject outliers in the input point cloud X, caused by the noisy segmentation mask M. In a nutshell, our robust corrector takes in the detected keypoints $\tilde{y}$, and produces a correction Δy^* by solving the *robust corrector optimization problem* (details below). The corrected keypoints then become:

$$\hat{y} = \tilde{y} + \Delta y^*. \qquad (4)$$

After computing the corrected keypoints $\hat{y}$, we can compute a *corrected pose* $\hat{T}(\Delta y^*)$ via outlier-free registration:

$$\hat{T}(\Delta y^*) = \arg\min_{T \in \text{SE}(3)} \ \sum_{i=1}^{N} \ \|\hat{y}[i] - T \cdot b[i]\|_2^2. \quad (5)$$

where N is the number of semantic keypoints.

We now describe the robust corrector optimization problem, and its forward and backward pass: since we are going to insert this module in a trainable architecture in Section V, we are particularly interested in making it differentiable.

Robust Corrector Optimization Problem. The robust corrector computes a correction term Δy, to the detected keypoints $\tilde{y}$, by attempting to match the input point cloud to the resulting posed CAD model $\hat{X}(\Delta y)$, which is arranged according to the corrected pose $\hat{T}(\Delta y^*)$. The corrector solves a bi-level optimization problem given by:

$$
\begin{aligned}
\underset{\Delta y \in \mathbb{R}^{3 \times N}}{\text{Minimize}} \quad & \frac{1}{n} \sum_{i=1}^{n} \rho\left(\min_j \ \|X[i] - \hat{X}(\Delta y)[j]\|_2\right), \\
\text{subject to} \quad & \hat{X}(\Delta y) = T(\Delta y) \cdot \hat{B}, \qquad (6) \\
& T(\Delta y) = \underset{T \in \text{SE}(3)}{\arg\min} \ \sum_{i=1}^{N} \|\tilde{y}[i] + \Delta y[i] - T \cdot b[i]\|_2^2
\end{aligned}
$$

where $X[i]$ and $\hat{X}(\Delta y)[j]$ denote the i-th and j-th points in the input X and posed CAD model $\hat{X}(\Delta y) = T(\Delta y) \cdot \hat{B}$, respectively. Here, $\hat{B}$ denotes a dense sampling of points on the CAD model B, and $\rho(\cdot)$ is a truncated least squares (TLS) robust loss function defined as $\rho(z) = \min\{z^2, \bar{c}^2\}$, where $\bar{c}$ is a user-specified threshold. In words, problem (6) computes the optimal keypoint correction such that the corrected keypoints produce a pose estimate $T(\Delta y)$ (via the last constraint in (6)) such that the CAD model posed at $T(\Delta y)$ (second constraint in (6)) matches the input (objective in (6)). The objective minimizes the distance between point pairs with distance below $\bar{c}$ and disregards points in the input X for which the closest point in $\hat{X}(\Delta y)$ is at a distance farther than $\bar{c}$. We use the TLS loss to automatically reject outliers in the input X, when solving the corrector optimization problem. The TLS

loss is commonly used in robust estimation for robotics [13], [45], [46] due to its insensitivity to outliers.

Forward Pass. The bi-level optimization problem (6) is non-linear and non-convex. However, we note that the lower-level optimization problem (last line in (6)) is nothing but an outlier-free registration problem, which can be solved in closed form via singular value decomposition (SVD) [47], [48]. Since the SVD computation is differentiable, we can directly compute the gradient of the cost function with respect to the correction $\Delta \boldsymbol{y}$. Therefore, we can solve the robust corrector problem (6) via a constant-step-size gradient descent. In our experiments, we implement the solver in PyTorch [49] —so as to enable batch processing— and compute the gradients using the autograd functionality in PyTorch.

Backward Pass. In order to include the robust corrector as a module of a trainable architecture, it is necessary to differentiate through the corrector optimization problem. We now show that the robust corrector optimization problem, although non-linear and non-convex, admits a very simple derivative of the output $\Delta \boldsymbol{y}^*$, with respect to the input $\tilde{\boldsymbol{y}}$.

Proposition 1. *The gradient of the correction $\Delta \boldsymbol{y}^*$ with respect to the detected keypoints $\tilde{\boldsymbol{y}}$ is the negative identity:*

$$\partial \Delta \boldsymbol{y}^* / \partial \tilde{\boldsymbol{y}} = -\mathbf{I}. \tag{7}$$

We omit the proof of the proposition, which remains identical to its non-robust counterpart in [9].

Remark 2 (Robust Corrector vs. Non-Robust Corrector vs. ICP). *The work [9] proposes a non-robust version of the corrector, that is recovered by using $\rho(z) = z^2$ in (6). The work also showed the advantage of optimizing over keypoints, rather than over poses (e.g., as in ICP). As we show in our experiments, the non-robust loss in [9] leads to large errors when the input point cloud contains outliers resulting from a noisy 2D segmentation. On the other hand, the proposed approach is able to correct large errors even in the presence of a large fraction of outliers.*

V. ENSEMBLE SELF-TRAINING ARCHITECTURE

Overview. Figure 4 illustrates the proposed ensemble self-training architecture. In the figure and description below, we limit the scope to stacking only two pose estimation models in parallel, but the methodology presented here can be trivially extended to a larger number of models (see Remark 4 below).

Let $\mathcal{M}^1$ and $\mathcal{M}^2$ be two pose estimators that take in $(\boldsymbol{I}, \boldsymbol{M}, \boldsymbol{X}, \boldsymbol{f})$ and output estimated pose and detected keypoints; below we will observe that keypoints detections can be artificially added to any pose estimator, as long as we have a CAD model of the object. We use $(\tilde{\boldsymbol{y}}^1, \tilde{\boldsymbol{T}}^1)$ and $(\tilde{\boldsymbol{y}}^2, \tilde{\boldsymbol{T}}^2)$ to denote the outputs produced by models $\mathcal{M}^1$ and $\mathcal{M}^2$, respectively. We augment each pose estimators with two modules: (i) robust corrector and (ii) observable correctness certificate check. The robust corrector takes in the estimated pose and detected keypoints $(\tilde{\boldsymbol{T}}, \tilde{\boldsymbol{y}})$, and outputs corrected pose and keypoints $(\hat{\boldsymbol{T}}^i, \hat{\boldsymbol{y}}^i)$, $i = 1, 2$, see Section IV.

The observable correctness certificate implements two checks to assess the geometric consistency of the corrected pose. The 3D certificate checks if the input point cloud $\boldsymbol{X}$ matches the posed CAD model $\hat{\boldsymbol{T}}^i \cdot \boldsymbol{B}$, for $i = 1, 2$. The 2D certificate, on the other hand, checks if the rendering of the posed CAD model, namely $\hat{\boldsymbol{T}}^i \cdot \boldsymbol{B}$, on the image plane is compatible with the input segmentation mask $\boldsymbol{M}$. Both tests account for occlusions and outliers in the detected object. These two certificates in turn determine if the corrected poses $\hat{\boldsymbol{T}}^1$ and $\hat{\boldsymbol{T}}^2$, produced by the robust correctors, are observably correct or not. The self-training uses the observably correct instances to supervise the training process. We now describe each of these modules, and the self-training method, in detail.

A. Robust Corrector Beyond Keypoint-based Models

In Section IV, we observed that a semantic-keypoint-based pose estimator processes the sensor data to obtain keypoint detections $\tilde{\boldsymbol{y}}$ and pose estimate $\tilde{\boldsymbol{T}}$. Therefore, we can think about such an estimator as a map from the input $(\boldsymbol{I}, \boldsymbol{M}, \boldsymbol{X}, \boldsymbol{f})$ to the tuple $(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{T}})$ of detected keypoints and pose estimate:

$$\mathcal{M} : (\boldsymbol{I}, \boldsymbol{M}, \boldsymbol{X}, \boldsymbol{f}) \rightarrow (\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{T}}), \tag{8}$$

where for the sake of generality we let $\mathcal{M}$ take $(\boldsymbol{I}, \boldsymbol{M}, \boldsymbol{X}, \boldsymbol{f})$ as input, whereas an estimator may use only a subset of the four inputs (*e.g.,* a point-cloud-only pose estimator may choose to work with only $\boldsymbol{X}$, in the four-tuple).

It is now easy to realize that if a pose estimator does not rely on keypoints and only computes a pose estimate $\tilde{\boldsymbol{T}}$, we can still hallucinate semantic-keypoint detections as

$$\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{T}} \cdot \boldsymbol{b}, \tag{9}$$

where $\boldsymbol{b}$ are the keypoints on the CAD model. Using $\tilde{\boldsymbol{y}}$ as keypoint detections, we can then apply the robust corrector optimization problem (6) to any pose estimator, as long as we have a CAD model $\boldsymbol{B}$ of the object we aim to detect.

B. Certificate of Observable Correctness

This section describes binary certificates that can distinguish correct estimates from incorrect ones, *i.e.,* a certificate that is equal to one when the estimate from a pose estimator is correct and zero otherwise. More precisely, we design certificates of *observable correctness* [9] that determines if the output produced by a learning-based model is consistent with the input data and the object CAD model. We propose two certificates of observable correctness, that assess consistency with both the 2D segmentation mask and the 3D point cloud.

3D Certificate. Let $\hat{\boldsymbol{T}}$ be the corrected pose produced by the robust corrector. We compute the posed CAD model $\hat{\boldsymbol{X}} = \hat{\boldsymbol{T}} \cdot \hat{\boldsymbol{B}}$, posed at $\hat{\boldsymbol{T}}$. Consider the set of distance-scores:

$$\mathbb{S}(\boldsymbol{X}, \hat{\boldsymbol{X}}) = \{ s_i = \min_j \| \boldsymbol{X}[i] - \hat{\boldsymbol{X}}[j] \|_2 \mid i = 1, \ldots, n \}. \tag{10}$$

The 3D certificate is then given by

$$\circ_{C3D}(\boldsymbol{X}, \hat{\boldsymbol{T}}) = \mathbb{I} \left\{ \texttt{percentile}(\mathbb{S}(\boldsymbol{X}, \hat{\boldsymbol{X}}), p) < \epsilon_{3D} \right\}, \tag{11}$$
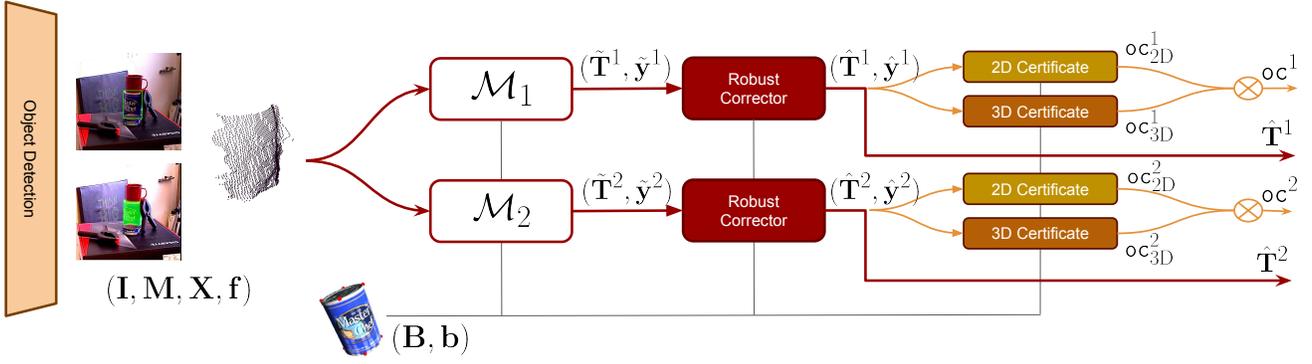
Fig. 4: **Ensemble self-training architecture.** The proposed architecture stacks several pose estimation models in parallel —we show only two for the sake of simplicity: model $\mathcal{M}^1$ and $\mathcal{M}^2$. The models take in the outputs produced by a 2D object detection and segmentation: $\boldsymbol{I}$ (scaled RoI of the detected object), $\boldsymbol{M}$ (2D segmentation mask of the detected object), $\boldsymbol{X}$ and $\boldsymbol{f}$ (point cloud of the detected object and color as point feature). The two models output object pose estimates and keypoints: $(\tilde{\boldsymbol{y}}^1, \tilde{\boldsymbol{T}}^1)$, $(\tilde{\boldsymbol{y}}^2, \tilde{\boldsymbol{T}}^2)$. The *robust corrector*, corrects the estimated poses, using dense information from the input $\boldsymbol{X}$ and the object CAD model $\boldsymbol{B}$. It produces corrected pose and keypoints: $(\hat{\boldsymbol{y}}^1, \hat{\boldsymbol{T}}^1)$, $(\hat{\boldsymbol{y}}^2, \hat{\boldsymbol{T}}^2)$. Two binary certificates, namely, the 2D and 3D certificates check geometric consistency of the corrected pose $\hat{\boldsymbol{T}}^1$ and $\hat{\boldsymbol{T}}^2$. If the 2D and 3D checks succeed, the corrected pose is declared to be *observably correct*. The architecture outputs an observably correct pose, and if none is found, it outputs the corrected pose of a preferred model, say pose $\hat{\boldsymbol{T}}^1$ from $\mathcal{M}^1$.

where $\texttt{percentile}(\mathbb{S}(\boldsymbol{X}, \hat{\boldsymbol{X}}), p)$ is the p-th percentile of the vector $\mathbb{S}(\boldsymbol{X}, \hat{\boldsymbol{X}})$ ($p = 0.9$ in our tests), and $\epsilon_{3\mathrm{D}}$ is a user-specified threshold. Intuitively, for $p = 0.9$, $\mathtt{oc_{3D}}(\boldsymbol{X}, \hat{\boldsymbol{T}}) = 1$ if at least 90% of the points in $\boldsymbol{X}$ is within a distance $\epsilon_{3\mathrm{D}}$ from a point in $\hat{\boldsymbol{X}}$, or $\mathtt{oc_{3D}}(\boldsymbol{X}, \hat{\boldsymbol{T}}) = 0$ otherwise. The percentile is used to account for outliers: taking a 90% percentile corresponds to assuming that the outliers $\boldsymbol{O_X}$ in (3) do not corrupt more than 10% of the points in the input $\boldsymbol{X}$.

2D Certificate. Let $\hat{\boldsymbol{T}}$ be the corrected pose produced by the robust corrector. We render the posed CAD model $\hat{\boldsymbol{T}} \cdot \boldsymbol{B}$ to obtain the mask $\hat{\boldsymbol{M}}$. Note that the $\hat{\boldsymbol{M}}$ will render the entire posed CAD model, hence will not account for occlusions, caused by other objects. In order to ensure that our 2D certificate is occlusion-aware, we compare the area of the detected mask $\boldsymbol{M}$ with the area of the intersection $\boldsymbol{M} \cap \hat{\boldsymbol{M}}$. The 2D certificate is then given by

$$\mathtt{oc_{2D}}(\boldsymbol{X}, \hat{\boldsymbol{T}}) = \mathbb{I}\left\{\frac{\mathrm{ar}\left(\boldsymbol{M} \cap \hat{\boldsymbol{M}}\right)}{\mathrm{ar}\left(\boldsymbol{M}\right)} > 1 - \epsilon_{2\mathrm{D}}\right\}, \quad (12)$$

where $\mathrm{ar}\left(\boldsymbol{M}\right)$ denotes the pixel area of all pixels (i, j) in the mask $\boldsymbol{M}$ with $\boldsymbol{M}(i, j) = 1$, and $\epsilon_{2\mathrm{D}}$ is a given threshold.

Observable Correctness Certificate. We say that an output $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{T}})$ is *observably correct* if the two checks —the 3D and 2D certificates— are met. More formally, we deem an output $(\hat{\boldsymbol{y}}, \hat{\boldsymbol{T}})$ to be observably correct when $\mathtt{oc}(\boldsymbol{X}, \hat{\boldsymbol{T}}) = 1$ with:

$$\mathtt{oc}(\boldsymbol{X}, \hat{\boldsymbol{T}}) = \mathtt{oc_{2D}}(\boldsymbol{X}, \hat{\boldsymbol{T}}) \cdot \mathtt{oc_{3D}}(\boldsymbol{X}, \hat{\boldsymbol{T}}). \quad (13)$$

In the next subsection, we show how to use the binary certificates to self-supervise an ensemble of pose estimators.

Remark 3 (Certificates vs Certifiable Correctness). *The work [9] shows that one can formally guarantee the correctness of a pose estimate (i.e., its proximity to the ground truth)* by computing a certificate of observable correctness and a certificate of non-degeneracy. The former evaluates how well the estimate fits the data; the latter ensures that the data contains enough information to compute a unique estimate. In this work, we only define certificates of observable correctness, hence we cannot derive formal performance guarantees. This is mostly due to the fact that we consider a realistic case with noisy segmentation, while the analysis [9] assumes perfect segmentation. Despite this theoretical gap, we empirically observe that the certificates of observable correctness provide an excellent tool for identifying good estimates.*

C. Ensemble Self-Training

We now describe our ensemble self-training procedure that trains multiple models in parallel, using just the binary-valued, observable correctness certificate to supervise.

Loss Functions. We first define two loss functions involved in our ensemble self-training. For a corrected $\hat{\boldsymbol{T}}$ and input $\boldsymbol{X}$, we define the self-supervised loss to be:

$$\mathcal{L}_{\mathrm{self}}\left(\boldsymbol{X}, \hat{\boldsymbol{T}}\right) = \frac{1}{n}\sum_i \rho\left(\min_j\|\boldsymbol{X}[i] - \hat{\boldsymbol{T}} \cdot \hat{\boldsymbol{B}}[j]\|_2\right), \quad (14)$$

where $\rho(z) = \min\{z^2, \bar{c}^2\}$ and $\bar{c}$ denotes the maximum admissible distance between pairs of points to be considered inliers. For a pose $\boldsymbol{T}'$, we also define a supervised loss to be

$$\mathcal{L}_{\mathrm{sup}}\left(\hat{\boldsymbol{T}}, \boldsymbol{T}'\right) = \frac{1}{m}\sum_i \min_j\|\hat{\boldsymbol{T}} \cdot \hat{\boldsymbol{B}}[i] - \boldsymbol{T}' \cdot \hat{\boldsymbol{B}}[j]\|_2^2$$

$$+ \frac{1}{m}\sum_j \min_i\|\hat{\boldsymbol{T}} \cdot \hat{\boldsymbol{B}}[i] - \boldsymbol{T}' \cdot \hat{\boldsymbol{B}}[j]\|_2^2, \quad (15)$$

where m denotes the number of points on the sampled point cloud $\hat{\boldsymbol{B}}$. In our definition in (15), we assume the loss to only back-propagate through $\hat{\boldsymbol{T}}$ (to train model weights), and not

through $\boldsymbol{T}'$. We remark that the supervised loss is nothing but the ADD-S [50] loss between the two posed models, namely $\hat{\boldsymbol{T}} \cdot \hat{\boldsymbol{B}}$ and $\boldsymbol{T}' \cdot \hat{\boldsymbol{B}}$; below we are going to use it in a way that each model in the ensemble supervises the others.

Self-Training. Recall that $\hat{\boldsymbol{T}}^1$ and $\hat{\boldsymbol{T}}^2$ denote the two poses outputted by model $\mathcal{M}^1$ and $\mathcal{M}^2$ and refined by the robust corrector (Fig. 4). Our self-training relies on stochastic gradient descent (SGD). At each SDG iteration and for each input $\boldsymbol{X}$ in a batch, we induce the following training loss:

$$
\begin{aligned}
\mathcal{L} = \circ\mathsf{c}(\boldsymbol{X}, \hat{\boldsymbol{T}}^1) \cdot \left[\mathcal{L}_{\text{self}}\left(\boldsymbol{X}, \hat{\boldsymbol{T}}^1 \right) + \mathcal{L}_{\text{sup}}\left(\hat{\boldsymbol{T}}^2, \hat{\boldsymbol{T}}^1 \right) \right] \\
+ \circ\mathsf{c}(\boldsymbol{X}, \hat{\boldsymbol{T}}^2) \cdot \left[\mathcal{L}_{\text{self}}\left(\boldsymbol{X}, \hat{\boldsymbol{T}}^2 \right) + \mathcal{L}_{\text{sup}}\left(\hat{\boldsymbol{T}}^1, \hat{\boldsymbol{T}}^2 \right) \right].
\end{aligned} \quad (16)
$$

The supervised loss $\mathcal{L}_{\text{sup}}\left(\hat{\boldsymbol{T}}^2, \hat{\boldsymbol{T}}^1 \right)$ only back-propagates through $\hat{\boldsymbol{T}}^2$, and trains model $\mathcal{M}^2$, and not $\mathcal{M}^1$. Similarly, $\mathcal{L}_{\text{sup}}\left(\hat{\boldsymbol{T}}^1, \hat{\boldsymbol{T}}^2 \right)$ back-propagates through $\hat{\boldsymbol{T}}^1$, and trains model $\mathcal{M}^1$, and not $\mathcal{M}^2$. Note that the loss only uses the input and the output of the different models but does not rely on external supervision. $\circ\mathsf{c}(\boldsymbol{X}, \hat{\boldsymbol{T}}^1)$ and $\circ\mathsf{c}(\boldsymbol{X}, \hat{\boldsymbol{T}}^2)$ are the binary-valued, observable correctness certificates. For instance, $\circ\mathsf{c}(\boldsymbol{X}, \hat{\boldsymbol{T}}^1) = 1$ indicates that the pose $\hat{\boldsymbol{T}}^1$ is observably correct. Therefore, whenever $\hat{\boldsymbol{T}}^1$ is observably correct, the loss (16) induces a self-supervised loss $\mathcal{L}_{\text{self}}\left(\boldsymbol{X}, \hat{\boldsymbol{T}}^1 \right)$ on model $\mathcal{M}^1$ and a supervised loss $\mathcal{L}_{\text{sup}}\left(\hat{\boldsymbol{T}}^2, \hat{\boldsymbol{T}}^1 \right)$ on model $\mathcal{M}^2$ (with supervision using $\hat{\boldsymbol{T}}^1$). Similarly, it induces a loss on $\mathcal{M}^1$ and $\mathcal{M}^2$, when $\hat{\boldsymbol{T}}^2$ is observably correct. This ensures that the observably correct outputs produced by one model are used to train the second model, and vice versa, taking advantage of the potential complementarity of the models in the ensemble.

Remark 4 (From Two to Many). *The training loss (16) is specified for training two models in parallel. This loss function can easily be extended to simultaneously train K pose estimation models. Let $\hat{\boldsymbol{T}}^1, \hat{\boldsymbol{T}}^2, \dots \hat{\boldsymbol{T}}^K$ denote the corrected poses output by the K pose estimators, after applying the robust corrector to each model. The training loss of our ensemble self-training—for each input $\boldsymbol{X}$— would then be:*

$$
\sum_{k=1}^{K} \circ\mathsf{c}(\boldsymbol{X}, \hat{\boldsymbol{T}}^k) \cdot \left[\mathcal{L}_{\text{self}}\left(\boldsymbol{X}, \hat{\boldsymbol{T}}^k \right) + \sum_{k' \neq k} \mathcal{L}_{\text{sup}}\left(\hat{\boldsymbol{T}}^{k'}, \hat{\boldsymbol{T}}^k \right) \right].
$$

Recall that the loss $\mathcal{L}_{\text{sup}}\left(\hat{\boldsymbol{T}}^{k'}, \hat{\boldsymbol{T}}^k \right)$ only back-propagates through $\hat{\boldsymbol{T}}^{k'}$ to train model $\mathcal{M}^{k'}$, and not $\mathcal{M}^k$.

Remark 5 (Role of the Robust Corrector). *The robust corrector is instrumental in extending the reach of each trained model. That is, it retains high accuracy for a much larger set of inputs, than what the pose estimation model —without the corrector— initially does. This allows bootstrapping the self-training process, i.e., ensures that during the initial self-training iterations there are enough observably correct inputs (which are then used for self-training) despite a potentially large domain gap; see results in Section VII-D.*

VI. OUTLIER-ROBUST POINT CLOUD PROCESSING

This section describes two improvements to existing regression-based architectures for keypoint detection; these improvements are mostly minor, but impact performance in practice, and are broadly applicable to a number of architectures, including point transformers [12] and point-net [10].

Challenges of Regression-based Architectures. The unstructured nature of point clouds makes it harder to device a simple pooling layer, that simultaneously samples representative points in the point clouds, and aggregates nearby features. Most point cloud-based architectures have resorted to using farthest point sampling strategy (FPS) [10], [11], [12], followed by aggregating features from the k-nearest neighbors. Such a pooling layer assumes that maximizing the distance between the sampled points is a good heuristic to select representative points in the point cloud. However, in the presence of outliers, FPS —by construction— tends to sample many outliers, instead of rejecting them. Furthermore, the lack of intrinsic translation invariance/equivariance also renders point-cloud-based models sensitive to the choice of a centroid, which is used in practice to center the point cloud and regain partial translation invariance/equivariance. To address these issues, we propose a *robust centroid* computation and *robust pooling*, which enhance the robustness of point-cloud-based models in the presence of outliers in the input points.

Robust Centroid. In the presence of outliers, the algebraic mean of the points in the point cloud $\boldsymbol{X}$ might be a poor proxy for the object center. We proposed to compute a *robust centroid* by solving a robust estimation problem:

$$
\bar{\boldsymbol{x}} = \arg\min_{\boldsymbol{u} \in \mathbb{R}^3} \frac{1}{n} \sum_i \rho\left(\|\boldsymbol{X}[i] - \boldsymbol{u}\|_2 \right), \quad (17)
$$

where $\rho(\cdot)$ is a robust loss function (*e.g.,* TLS). We solve (17) using the graduated non-convexity algorithm [13].

The input point cloud is then centered at $\bar{\boldsymbol{x}}$, namely,

$$
\boldsymbol{X} \leftarrow \boldsymbol{X} - \bar{\boldsymbol{x}} \otimes \mathbf{1}_n^\mathsf{T} \quad (18)
$$

where $\mathbf{1}_n$ is the vector of ones, before passing it to the point-cloud architecture. If the output is desired to be translation equivariant, the robust centroid $\bar{\boldsymbol{x}}$ is also added to the model output. We remark that the robust centroid computation does not require to be differentiable, since it is applied directly to the input and there are no trainable weights in (17).

Robust Pooling. Pooling is a building block of existing keypoint detection architectures, see [10], [11], [12]. We propose a simple, trainable pooling layer, which samples points based on regressed scores for each point. The pooling layer takes an input point cloud with features $(\boldsymbol{X}, \boldsymbol{f}) \in \mathbb{R}^{3 \times n} \times \mathbb{R}^{d \times n}$ and outputs another point cloud, with features $(\boldsymbol{X}', \boldsymbol{f}') \in \mathbb{R}^{3 \times n'} \times \mathbb{R}^{d \times n'}$; d denotes the feature dimension ($d = 3$ in our pose estimation setup), and $n' < n$ since the number of pooled points is always smaller than the number of input points.

Given the input features $\boldsymbol{f} = [\boldsymbol{f}_1, \dots \boldsymbol{f}_n]$, we first regress a score $s_i = \text{MLP}(\boldsymbol{f}_i)$ for each point i using a multilayer perceptron $\text{MLP}(\cdot)$. We select n' ($< n$) points in $(\boldsymbol{X}, \boldsymbol{f})$ that

have the highest scores to obtain $(\boldsymbol{X}', \boldsymbol{f}')$. In our experiments, we implement MLP $(\cdot)$ to have one hidden layer and ReLU activations. The trainable MLP $(\cdot)$ weights make the layer malleable during training to automatically learn to reject outliers in the input point cloud.

Remark 6 (Robust Pooling as Trainable Sampling)**.** *The robust pooling layer only samples points and does not aggregate features from nearby points to create new features — as a traditional pooling layer does. We assume here that other blocks in the architecture will learn to extract the necessary features, before pooling. Treating pooling as trainable sampling not only simplifies the architecture, but also saves compute time that is otherwise spent in performing k-nearest-neighbor search and aggregation operation.*

VII. Experiments

We present four sets of experiments. We first demonstrate the effectiveness of the robust corrector in correcting large keypoint errors (Section VII-A), and show its utility viz-a-viz the non-robust corrector proposed in [9] (Section VII-B). We then show the ability of the robust centroid and pooling strategies, developed in Section VI, in mitigating the effects of outliers during keypoint detection (Section VI). Finally, we show the effectiveness of our ensemble self-training architecture by self-training a point-cloud-based model and a CNN-based model (CosyPose [8]) in parallel (Section VII-D).

A. Robust Corrector Analysis

Setup. We use the YCBV dataset and objects [51], [7]. For each YCBV object, we extract the object depth point cloud from the YCBV test set. In this subsection, we use the ground-truth pose annotations to get the ground-truth keypoint annotations using (9), and we set the detected keypoints $\tilde{\boldsymbol{y}}$ to be ground-truth semantic keypoints plus an additive perturbation. If $\boldsymbol{y}^*$ denotes the ground-truth keypoints, our noise model adds uniform noise in range $[-\sigma D/2, \sigma D/2]$, to keypoint $\boldsymbol{y}^*[i]$, with probability f, where D is the object diameter; this allows us to simulate increasing levels of keypoint detection errors by increasing σ. We set $f = 0.8$ and analyze the robust corrector as a function of the noise variance parameter σ.

Results. The robust corrector is designed to correct keypoint detection errors. Fig. 5(a) plots the normalized ADD-S (*i.e.,* the ADD-S normalized by the object diameter D) as a function of the keypoint detection noise parameter σ. We plot these scores for the robust corrector, as well as the observably correct instances produced by the robust corrector. We evaluate against two baselines: (i) Naive: which sets $\hat{\boldsymbol{y}} = \tilde{\boldsymbol{y}}$ in solving the outlier-free registration problem (5), and (ii) Naive + ICP: which uses outlier-free registration (same as Naive) and then refines the result with point-to-point ICP. The threshold $\bar{c}$ for the robust corrector and the maximum points-pair correspondence distance in ICP are both set to be 30% of the object diameter. Fig. 5(b) plots the fraction of observably correct instances produced by each method, as a function of σ.

Insights. We observe that the robust corrector is able to correct large keypoint errors, leading to improved pose estimates (Fig. 5(a)). Even when $\sigma = 0.6$, *i.e.,* when 80% of the keypoints are perturbed by uniform noise proportional to 60% of the object diameter, we see that around 80% of the outputs produced by the robust corrector are observably correct (Fig. 5(b)), and the observably correct instances are highly accurate (dashed red line in Fig. 5(a)). This number drops to 0% for Naive (Fig. 5(b)), which indeed exhibits much larger errors (Fig. 5(a)). Naive + ICP outperforms Naive, in terms of accuracy and fraction of observably correct instances. The robust corrector outperforms both — especially for large noise (σ) values.

B. Robust versus Non-Robust Corrector

Setup. We implement the non-robust corrector proposed in [9], which solves (6), but with $\rho(z) = z^2$. We compare the latter against the proposed robust corrector in the presence of increasing number of outlier points in the input $\boldsymbol{X}$. We use a setup similar to Section VII-A, but now we fix $\sigma = 0.4$ (along with $f = 0.8$), and add outliers to $\boldsymbol{X}$. We evaluate the ADD-S score as a function of the outlier rate: for instance, when the outlier rate is 0.5, we test on a point cloud where 50% of the points have been replaced with random points.

Results and Insights. Figure 5(c) plots the normalized ADD-S score as a function of the outlier rate. We observe that the proposed robust corrector significantly outperforms the non-robust corrector from [9]. We also observe that, in the presence of outliers, the non-robust corrector yields worse performance than having no corrector at all, which is expected, since the non-robust corrector will be increasingly biased by the outliers in the point cloud $\boldsymbol{X}$.

C. Impact of Robust Centroid and Robust Pooling

Robust Keypoint Network (RKN). To evaluate the effect of the robust centroid and robust pooling, we integrate these two modules into a point transformer architecture [12] for semantic keypoint detection. In particular, we subtract the robust centroid to the point cloud before passing it to the detector. Then, in the architecture, we alternate each point transformer block [12] (which extracts and transforms features), with a robust pooling layer (that sparsifies the point cloud). This is repeated several times, and the final result is passed to a multi-layered perceptron, which regresses the keypoints $\tilde{\boldsymbol{y}}$. Finally, the centroid is added back to the keypoints to regain translation equivariance. We call the resulting architecture *Robust Keypoint Network* (RKN): RKN regresses semantic keypoints $\tilde{\boldsymbol{y}}$, given a point cloud with color as point features.

Setup. We evaluate RKN against baselines that use the same architecture as RKN, but without the robust pooling and centroid; we refer to those with the label "KeyPo". In particular, we consider the following variants: (i) KeyPo (FPS): which is KeyPo with farthest point sampling for pooling, (ii) KeyPo (Random): which is KeyPo with random point sampling for pooling, (iii) KeyPo (No Pooling): which is KeyPo without any pooling layers, (iv) KeyPo (Robust Pooling): which is
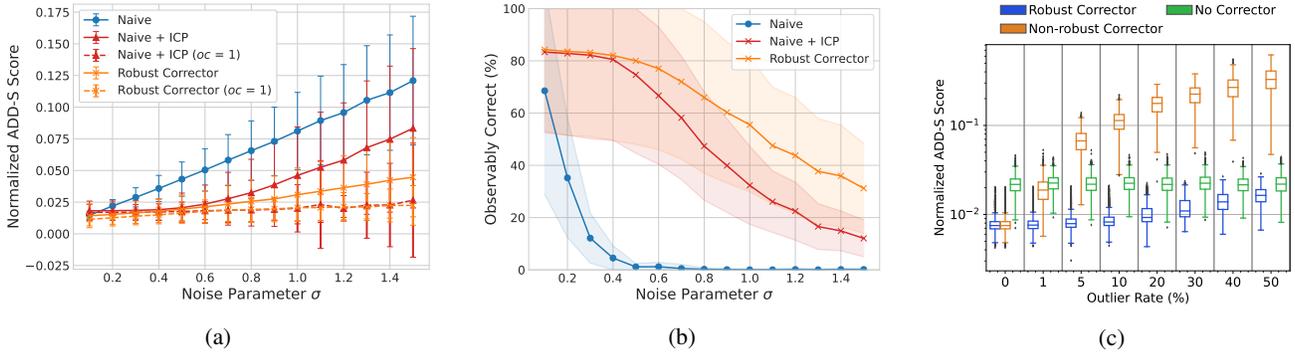
Fig. 5: (a) Normalized ADD-S (averaged across all objects in the YCBV dataset) and corresponding standard deviation (shown as error bars) as a function of the noise parameter σ. (b) Fraction of observably correct instances ($\mathrm{oc} = 1$) (averaged across all objects in the YCBV dataset) as a function the noise parameter σ. (c) ADD-S as a function of the outlier rate.
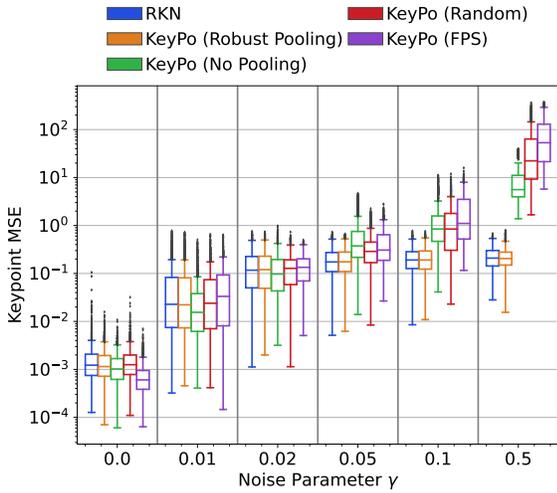

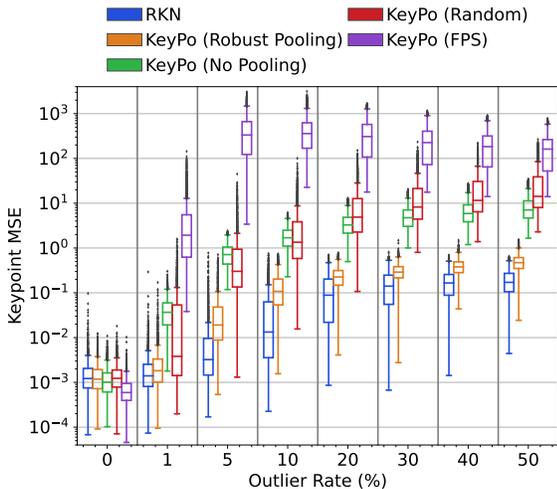Fig. 6: Mean squared error (MSE) in the keypoint detection as a function of the added noise variance γ.

Fig. 7: Mean squared error (MSE) in the keypoint detection as a function of the outlier rate in the input $\boldsymbol{X}$.

KeyPo with the proposed robust pooling, and finally, (v) RKN, which is nothing but KeyPo with the proposed robust centroid and robust pooling. We remark that KeyPo (FPS), KeyPo (Random), KeyPo (No Pooling), and KeyPo (Robust Pooling) use a non-robust centroid computation. We use the YCBV dataset [51], [7] and train the keypoint detectors in a fully supervised manner. We evaluate the trained models in two settings. The first adds zero-mean Gaussian noise, with standard deviation γ, to each point in the input $\boldsymbol{X}$. The second adds outlier points to the input $\boldsymbol{X}$.

Results and Insights. Fig. 6 plots the mean squared error (MSE) in the detected keypoints as a function of the added noise standard deviation γ; Fig. 7 plots the MSE of the detected keypoints as a function of the outlier rate in $\boldsymbol{X}$. We observe that RKN, with robust centroid and robust pooling, outperforms all the baselines. We see that while KeyPo (Robust Pooling) shows performance competitive to RKN in the case of added noise but no outliers (Fig. 6), it does not fare well in the presence of outliers. We also observe that popular pooling methods, such as farthest point sampling and random sampling, tend to do worse than using no sampling at all.

D. The YCBV and TLESS Experiment

Setup. We consider an ensemble self-training architecture with two complementary pose estimators: CosyPose [8], a CNN-based, RGB-only pose estimator, and RKN, the point cloud-based pose estimator described in Section VII-C. We take sim-trained CosyPose and RKN, and stack them in our ensemble self-training architecture and self-train them on a real unannotated dataset, using the self-training method in Section V. We label the resulting self-trained model "Ensemble (SSL)" (as in self-supervised learning); we also report the accuracy achieved only by the outputs deemed observably correct: we denote the resulting results as "Ensemble (SSL, $\mathrm{oc} = 1$)". Moreover, we also evaluate the performance of each branch in the ensemble self-training architecture. Ensemble-CosyPose denotes the results corresponding to the corrected poses from the CosyPose branch. Similarly, Ensemble-RKN denotes the results corresponding to the corrected poses
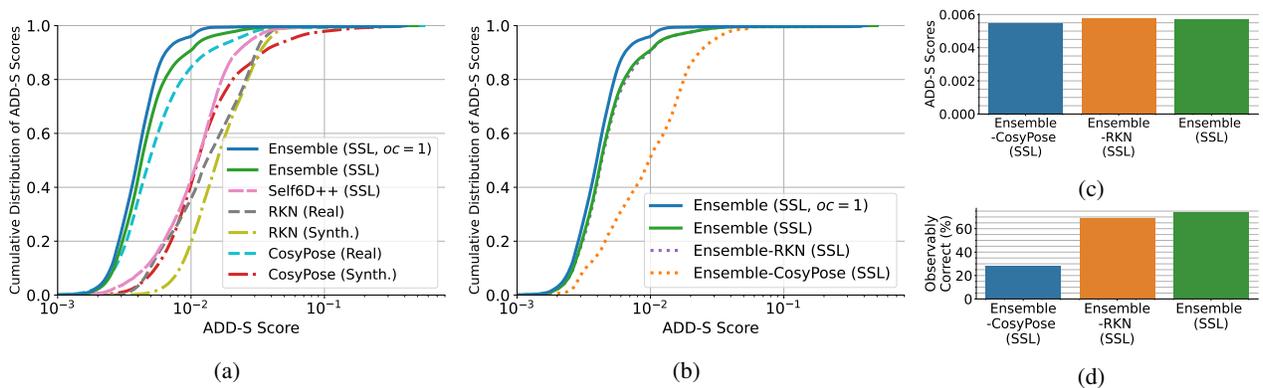
Fig. 8: (a) Cumulative distribution of ADD-S scores, averaged across all objects in the YCBV dataset, for the proposed ensemble self-training architecture (Ensemble) and other baselines. (b) Cumulative distribution of ADD-S scores, across all objects in the YCBV dataset, for Ensemble and its branches. (c) Average ADD-S score attained by the observably correct outputs of Ensemble and its branches. (d) Percentage of observably correct instances for Ensemble and its branches.

from the RKN branch. We compare these results against fully supervised versions of CosyPose and RKN, and against Self6D++ [15], a state-of-the-art self-supervised method, with weights and hyper-parameters provided by the authors.

We train the supervised baselines on the YCBV and T-LESS training set in the BOP dataset [7]; we use the given training and test split. For ensemble self-training, the initial models are pre-trained on the synthetic data generated by a rendering engine, also provided in the BOP dataset. The models are then self-trained on the real test dataset. We use the stochastic gradient descent optimizer, with a learning rate of $2 \cdot 10^{-2}$ for RKN and $3 \cdot 10^{-4}$ for CosyPose for 20 epochs over the standard test splits. Other hyper-parameters are reported in Appendix C. For the supervised baselines, we use the suffix "Synth." when they are trained on the synthetic data, or "Real" when they are trained on the real training data; for instance "CosyPose (Real)" corresponds to the state-of-the-art approach from [8], which is trained in a supervised manner using the training data from YCBV or TLESS. For all approaches we use the 2D segmentation masks provided by the datasets (Fig. 2). We measure performance using the ADD-S score [50], computed on the test set and averaged across all objects.

Results and Insights. Fig. 8(a) shows the cumulative distribution of the ADD-S scores for each approach. The figure shows that the proposed ensemble self-training outperforms by a small margin the fully supervised RKN (Real) and CosyPose (Real) approaches on the YCBV test dataset. This shows that the ensemble self-training architecture is able to reap the complementary benefits provided by the two models, and enhance them, without the need for external supervision. The figure also shows the performance of only the observably correct outputs produced by Ensemble, *i.e.,* Ensemble (SSL, oc = 1). We see that these exhibit a further performance boost, indicating that our certificates of observable correctness are indeed able to identify correct outputs. The figure also shows that RKN (Synth.) and CosyPose (Synth.), which are trained on synthetic data, perform poorly, indicating a large

sim-to-real gap, and remarking the capability of the proposed ensemble self-training to self-train starting from poor initial models. Finally, we observe that the proposed ensemble self-training significantly outperforms Self6D++ — a state-of-the-art, self-supervised pose estimation method [15].

Figure 8(b) shows how each branch in the ensemble self-training architecture contributes to its overall performance. We observe that Ensemble-RKN contributes the most. This is partly because RKN, in the RKN-branch, works directly on the RGB-D input, as opposed to CosyPose, which only relies on RGB information. However, a significant performance boost to RKN is provided by the robust corrector (*cf.* with Fig. 8(a), where RKN (Real) does much worse than CosyPose (Real)).

While Fig. 8(a)-(b) showed that the observably correct outputs correspond to highly accurate pose estimates —this is also true for each branch, see Fig. 8(c)— not all outputs produced by Ensemble are observably correct. Figure 8(d) shows the percentage of observably correct outputs produced by Ensemble, and each of its branches. We observe a gap in terms of the % of observably correct instances produced. While the RKN-branch produces 67% observably correct outputs, this number is 28% for the CosyPose branch. The overall Ensemble architecture, as expected, gets the best of both models, and 70% of its outputs are observably correct.

Tables I and II provide further insights into the compared techniques by breaking down the results by objects (Fig. 8 instead averaged results across all objects). The tables report the threshold ADD-S score with a threshold equal to 5% of the object diameter, and ADD-S (AUC) with a threshold equal to 10% of the object diameter. Note that this is a much stricter setup, compared to [15], which uses a much larger ADD-S threshold of 10 cm. Table I shows the performance on six YCBV objects, and Table II on six T-LESS objects; evaluation of all YCBV and TLESS objects is given in Tables V and VI. We see that our conclusions, gleaned from Fig. 8, still hold when parsing the object-specific performance. This reinforces that the proposed ensemble self-training ensures very accurate pose estimates and performs on par or better

than fully supervised baselines while not requiring real-world 3D annotations. Qualitative results comparing our Ensemble against the supervised baselines, RKN (Real) and CosyPose (Real), are given in Fig. 1.

Insights: Impact of Corrector. The robust corrector in Section IV was proposed to help bridge the sim-to-real gap. Table III validates our proposal by showing the threshold ADD-S scores for the sim-trained models, RKN (Synth.) and CosyPose (Synth.), with and without the robust corrector, along with the fully supervised models on the real data, *i.e.,* RKN (Real) and CosyPose (Real). We see that the robust corrector provides the anticipated performance boost, and helps to bridge the sim-to-real gap. Extra results with cumulative distributions of ADD-S scores are given in Appendix A.

Insights: Progression of the Ensemble Self-Training. Figure 1 shows that the number of observably correct instances grows as the ensemble self-training progresses. Appendix B provides further results, showing the increase in the average number of observably correct instances for both branches of the ensemble, as well as breaking down the results in terms of 2D and 3D certificates.

VIII. CONCLUSION

We advance self-supervised learning for object pose estimation by proposing an ensemble self-training architecture that simultaneously trains multiple models without manual 3D annotations, and leverages the complementarity of the models to further boost their performance. The ensemble self-training architecture is enabled (i) by a differentiable robust corrector, which refines the pose estimates by each model, and (ii) by the definition of observable correctness certificates that identify correct pose estimates at test-time. The proposed ensemble self-training performs on par or better compared to state-of-the-art, fully supervised methods and largely outperforms competing self-supervised baselines. As additional contributions, we introduce a robust centroid computation and robust pooling operation that empirically enhance the performance of point-cloud-based architectures for keypoint detection in the presence of outliers caused by a noisy 2D object segmentation.

The results in this paper open several avenues of future work. First, it would be interesting to extend the proposed self-supervised approach to category-level perception where the object CAD model is unknown or has to be selected from a library of CAD models [42]. Second, it would be interesting to enhance the robust corrector to leverage 2D information (currently, it only uses the point cloud). Finally, the proposed robust centroid and robust pooling are expected to be useful for other point-cloud processing tasks, including point cloud segmentation and shape completion.

APPENDIX

A. Impact of the Robust Corrector

The proposed robust corrector is designed to help bridge the sim-to-real gap. Table III in the main paper validates our proposal by showing the threshold ADD-S scores for the sim-trained models, RKN (Synth.) and CosyPose (Synth.),

(a)

(b)
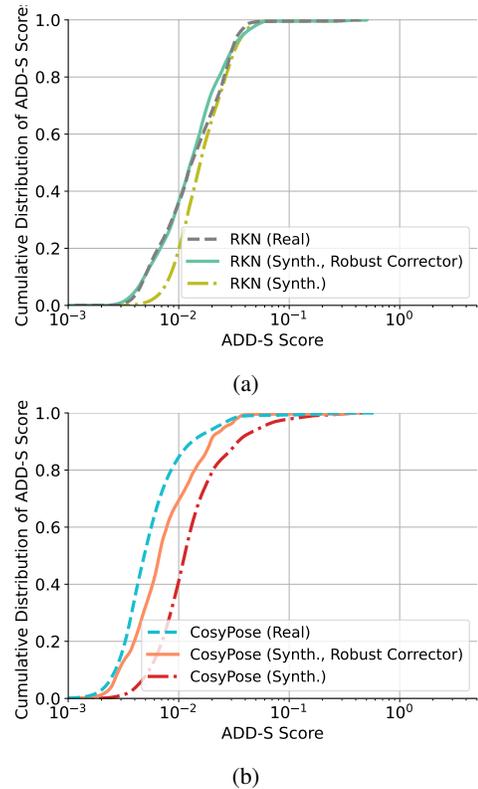
Fig. 9: Cumulative distribution of ADD-S scores, averaged across all objects in the YCBV dataset. (a) The robust Corrector bridges the sim-to-real gap for RKN. (b) The robust Corrector partially bridges the sim-to-real gap for CosyPose.

with and without the robust corrector, along with the fully supervised models on the real data, *i.e.,* RKN (Real) and CosyPose (Real). Figure 9 plots the full distribution of the ADD-S scores for these models, further confirming that the corrector does indeed help bridge the sim-to-real gap.

B. Progression of the Ensemble Self-Training

Figure 1 in the main paper showed that the number of observably correct instances grows as the ensemble self-training progresses, confirming the effectiveness of the proposed self-training. Figure 10 provides further insights by showing the average increase in the percentage of instances with ($\circ c_{2D} = 1$) and ($\circ c_{3D} = 1$), produced by both the branches, after our ensemble self-training. For CosyPose, we also consider a variant that implements a trainable coarse detector, which directly regresses object pose, while the CosyPose (Refine) implements a pre-trained coarse detector along with a trainable pose refinement model [52]. CosyPose (Refine), in fact, is what is proposed in [8] to achieve high accuracy. We observe that the RKN achieves the highest increase: 23% for ($\circ c_{2D} = 1$) and 56% for ($\circ c_{3D} = 1$). We also see an increase for the CosyPose branch. The CosyPose (Coarse) shows a higher improvement (10% and 24%) compared to CosyPose (Refine) (1% and 3%). Despite the modest increase in CosyPose (Refine), it still shows higher pose estimation accuracy after

TABLE I: Evaluation of Ensemble and baselines on the YCBV dataset.

ADD-S ADD-S (AUC)	Coffee Can		Sugar Box		Tuna Can		Wood Block		Scissors		Large Clamp	
RKN (Real)	0.90	0.65	0.14	0.16	0.62	0.50	0.05	0.18	0.01	0.16	0.12	0.23
CosyPose (Real)	0.83	0.63	1.00	0.81	0.93	0.70	0.27	0.29	0.22	0.28	0.86	0.67
Self6D++ (SSL)	0.25	0.29	0.36	0.43	0.32	0.33	0.26	0.25	0.11	0.14	0.38	0.32
Ensemble (SSL)	1.00	0.77	0.99	0.79	1.00	0.76	0.98	0.69	0.96	0.73	0.97	0.78
Ensemble (SSL, $\mathtt{oc}=1$)	1.00	0.78	1.00	0.83	1.00	0.79	1.00	0.74	1.00	0.77	0.97	0.78

TABLE II: Evaluation of Ensemble and baselines on the T-LESS dataset.

ADD-S ADD-S (AUC)	obj_000004		obj_000010		obj_000013		obj_000024		obj_000026		obj_000030	
RKN (Real)	0.19	0.32	0.62	0.52	0.46	0.48	0.31	0.39	0.40	0.47	0.79	0.58
CosyPose (Real)	0.41	0.38	0.84	0.63	0.51	0.42	0.62	0.50	0.83	0.64	0.93	0.71
Ensemble (SSL)	0.38	0.42	0.74	0.56	0.43	0.46	0.66	0.52	0.53	0.52	0.97	0.73
Ensemble (SSL, $\mathtt{oc}=1$)	0.85	0.65	0.99	0.73	0.79	0.59	0.98	0.69	1.00	0.72	1.00	0.75

TABLE III: Effects of the robust corrector on closing the sim-to-real gap for both CosyPose and RKN.

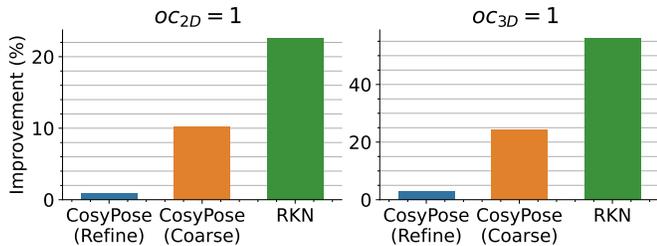| ADD-S	Synth.	Synth. with Robust Corrector	Real
CosyPose	0.41	0.69	0.85
RKN	0.17	0.36	0.36

Fig. 10: Average percentage increase in the number of instances with ($\mathtt{oc_{3D}}=1$) and ($\mathtt{oc_{2D}}=1$), after ensemble self-training. Results are for the RKN branch, and two variations of the CosyPose branch, namely, CosyPose (Coarse) and CosyPose (Refine).

self-training, compared to CosyPose (Coarse). Therefore, in the baseline comparisons we only show CosyPose (Refine).

C. List of Hyper-Parameters

The proposed ensemble self-training used stochastic gradient descent optimizer with a learning rate of $2 \cdot 10^{-2}$ for RKN and $3 \cdot 10^{-4}$ for CosyPose. The momentum and weight decay was set to 0.9 and $1 \cdot 10^{-5}$, respectively, and a batch size of 20 was used during the ensemble self-training. The clamp threshold $\bar{c}$ in the robust corrector (eq. (6) in the main paper) and the loss function (eq. (14) in the main paper) was set to 10% of the object diameter. The $\epsilon_{\mathtt{oc_{3D}}}$ and $\epsilon_{\mathtt{oc_{2D}}}$, used in the 3D and 2D certificates (eqs. (11)-(12) in the main paper), were tuned manually for different objects. For YCBV, $\epsilon_{\mathtt{oc_{3D}}}$ was set to 4% of the object diameter, whereas $\epsilon_{\mathtt{oc_{2D}}}$ was chosen to be the best among: 50%, 60%, and 95% of the object diameter. For TLESS, $\epsilon_{\mathtt{oc_{2D}}}$ was set to 90% of the object diameter, whereas $\epsilon_{\mathtt{oc_{3D}}}$ was chosen to be the best among: 4%, 6%, and 8% of the object diameter. We had to be lenient in the choice

of $\epsilon_{\mathtt{oc_{2D}}}$ because of the inaccuracies in the detected masks. Table IV lists them for all the YCBV and TLESS objects.

D. Additional Visualization

Figure 11 shows some examples of pose estimates resulting from the proposed Ensemble and other baseline methods.

REFERENCES

[1] K. Ebadi, L. Bernreiter, H. Biggie, G. Catt, Y. Chang, A. Chatterjee, C. Denniston, S.-P. Deschênes, K. Harlow, S. Khattak, L. Nogueira, M. Palieri, P. Petráček, P. Petrlík, A. Reinke, V. Krátký, S. Zhao, A. Agha-mohammadi, C. Alexis, C. Heckman, K. Khosoussi, N. Kottege, B. Morrell, M. Hutter, F. Pauling, F. Pomerleau, M. Saska, S. Scherer, R. Siegwart, J. Williams, and L. Carlone, "Present and future of SLAM in extreme underground environments," *arXiv preprint: 2208.01787*, 2022.

[2] A. Zeng, K. T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao, "Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1386–1383.

[3] B. Chen, J. Cao, A. Parra, and T.-J. Chin, "Satellite pose estimation with deep landmark regression and nonlinear pose refinement," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 2019, pp. 2816–2824.

[4] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 36–52, Sep. 2015.

[5] T. Hodan, P. Haluza, S. Obdrzalek, J. Matas, M. Lourakis, and X. Zabulis, "T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-less Objects," in *IEEE Winter Conf. Appl. Computer Vision (WACV)*, Mar. 2017.

[6] T. Hodaň, F. Michel, E. Brachmann, W. Kehl, A. G. Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T.-K. Kim, J. Matas, and C. Rother, "BOP: Benchmark for 6D Object Pose Estimation," in *European Conf. on Computer Vision (ECCV)*, 2018, pp. 19–35.

[7] T. Hodaň, M. Sundermeyer, B. Drost, Y. Labbé, E. Brachmann, F. Michel, C. Rother, and J. Matas, "BOP challenge 2020 on 6D object localization," *European Conference on Computer Vision Workshops (ECCV)*, 2020.

[8] Y. Labbe, J. Carpentier, M. Aubry, and J. Sivic, "CosyPose: Consistent multi-view multi-object 6D pose estimation," in *European Conf. on Computer Vision (ECCV)*, 2020.

[9] R. Talak, L. Peng, and L. Carlone, "Certifiable 3D object pose estimation: Foundations, learning models, and self-training," *IEEE Trans. Robotics*, (to appear) 2023, arXiv preprint: 2206.11215 (pdf).

[10] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3D classification and segmentation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 652–660.
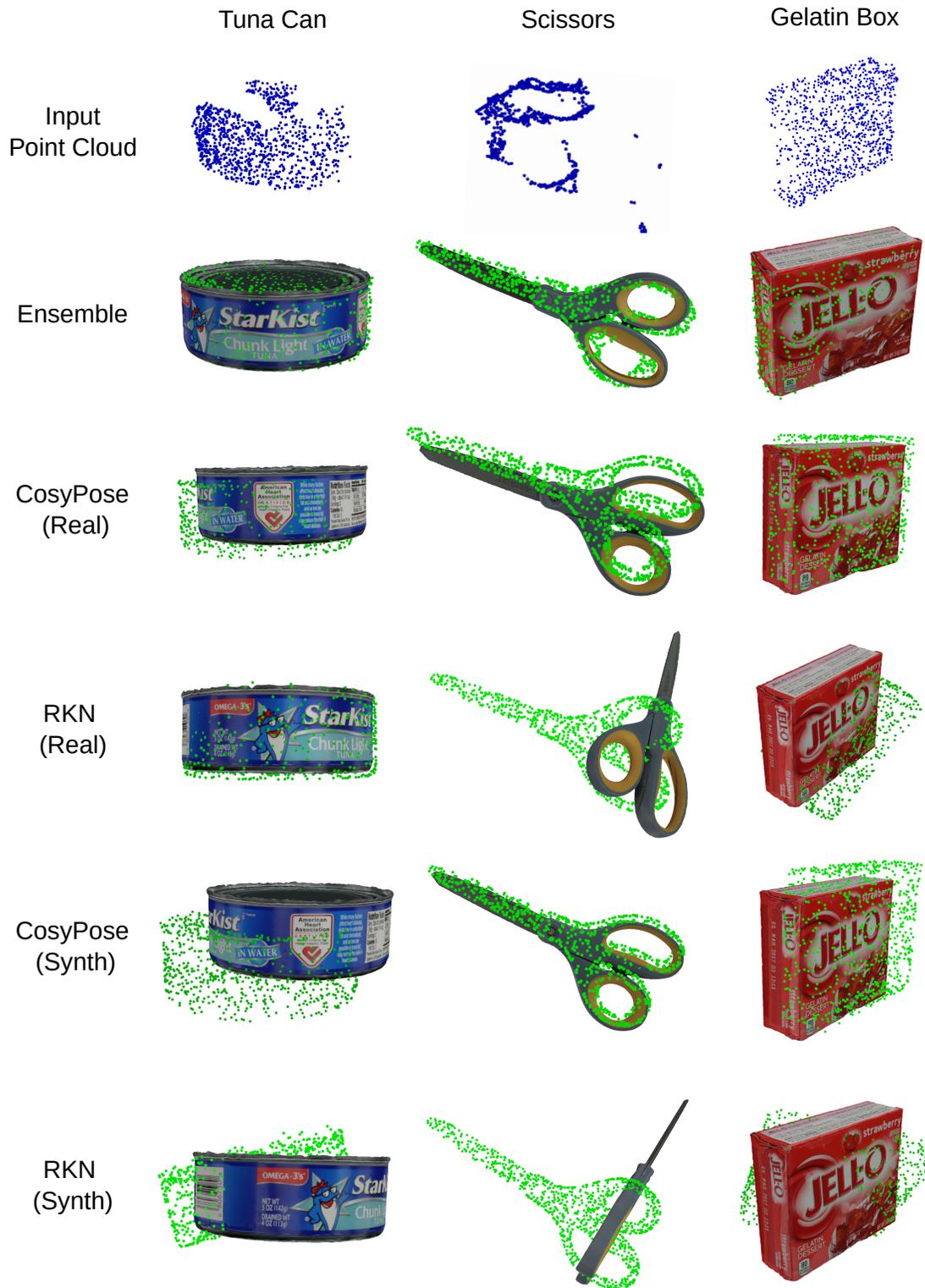
Fig. 11: Sample estimates of the proposed ensemble self-training (label: Ensemble) and various baseline methods on three objects from the YCBV dataset. The first row displays input point clouds generated from depth maps using segmentation masks. Green dots represent model point clouds transformed with ground-truth transformations, and the CAD models are transformed using the pose estimates.

TABLE IV: Parameters $\epsilon_{\mathrm{oc_{3D}}}$ and $\epsilon_{\mathrm{oc_{2D}}}$, as a % of the object diameter, used for the YCBV and TLESS objects.

	object	$\epsilon_{\mathrm{oc_{3D}}}$	$\epsilon_{\mathrm{oc_{2D}}}$	object	$\epsilon_{\mathrm{oc_{3D}}}$	$\epsilon_{\mathrm{oc_{2D}}}$	object	$\epsilon_{\mathrm{oc_{3D}}}$	$\epsilon_{\mathrm{oc_{2D}}}$	object	$\epsilon_{\mathrm{oc_{3D}}}$	$\epsilon_{\mathrm{oc_{2D}}}$
YCBV	obj_01	4%	95%	obj_02	4%	95%	obj_03	4%	95%	obj_04	4%	95%
	obj_05	4%	60%	obj_06	4%	95%	obj_07	4%	50%	obj_08	4%	95%
	obj_09	4%	60%	obj_10	4%	50%	obj_11	4%	95%	obj_12	4%	60%
	obj_13	4%	95%	obj_14	4%	95%	obj_15	4%	50%	obj_16	4%	95%
	obj_17	4%	50%	obj_18	4%	50%	obj_19	4%	50%	obj_20	4%	50%
	obj_21	4%	60%	–			–			–		

	object	$\epsilon_{\mathrm{oc_{3D}}}$	$\epsilon_{\mathrm{oc_{2D}}}$	object	$\epsilon_{\mathrm{oc_{3D}}}$	$\epsilon_{\mathrm{oc_{2D}}}$	object	$\epsilon_{\mathrm{oc_{3D}}}$	$\epsilon_{\mathrm{oc_{2D}}}$	object	$\epsilon_{\mathrm{oc_{3D}}}$	$\epsilon_{\mathrm{oc_{2D}}}$
TLESS	obj_01	4%	90%	obj_02	4%	90%	obj_03	4%	90%	obj_04	4%	90%
	obj_05	6%	90%	obj_06	6%	90%	obj_07	6%	90%	obj_08	6%	90%
	obj_09	6%	90%	obj_10	6%	90%	obj_11	6%	90%	obj_12	6%	90%
	obj_13	4%	90%	obj_14	4%	90%	obj_15	8%	90%	obj_16	8%	90%
	obj_17	4%	90%	obj_18	4%	90%	obj_19	6%	90%	obj_20	6%	90%
	obj_21	6%	90%	obj_22	6%	90%	obj_23	6%	90%	obj_24	4%	90%
	obj_25	4%	90%	obj_26	4%	90%	obj_27	6%	90%	obj_28	6%	90%
	obj_29	4%	90%	obj_30	4%	90%	–			–		

TABLE V: Evaluation of Ensemble and baselines on 21 YCBV objects.

ADD-S	ADD-S (AUC)	Coffee Can		Cracker Box		Sugar Box		Soup Can		Mustard Bottle		Tuna Can	
RKN (Real)		0.90	0.65	0.01	0.06	0.14	0.16	0.64	0.51	0.64	0.47	0.62	0.50
CosyPose (Real)		0.83	0.63	**0.98**	**0.75**	**1.00**	**0.81**	0.85	0.65	1.00	0.78	0.93	0.70
Self6D++		0.25	0.29	0.77	0.63	0.36	0.43	0.09	0.17	0.80	0.67	0.32	0.33
Ensemble (SSL)		**1.00**	**0.77**	0.68	0.55	0.99	0.79	**0.96**	**0.74**	**1.00**	**0.78**	**1.00**	**0.76**
Ensemble (SSL, oc = 1)		1.00	0.78	0.96	0.75	1.00	0.83	0.98	0.76	1.00	0.78	1.00	0.79

ADD-S	ADD-S (AUC)	Pudding Box		Gelatin Box		Meat Can		Banana		Pitcher		Bleach	
RKN (Real)		0.01	0.18	0.00	0.15	0.40	0.35	0.18	0.23	0.70	0.55	0.39	0.35
CosyPose (Real)		0.91	0.79	0.88	0.70	0.61	0.51	0.94	0.77	**1.00**	**0.84**	0.82	0.67
Self6D++		0.63	0.58	0.05	0.16	0.30	0.30	0.90	0.68	1.00	0.77	0.77	0.62
Ensemble (SSL)		**1.00**	**0.80**	**1.00**	**0.85**	**0.71**	**0.59**	**1.00**	**0.79**	0.81	0.66	**1.00**	**0.76**
Ensemble (SSL, oc = 1)		1.00	0.80	1.00	0.85	0.80	0.65	1.00	0.79	1.00	0.81	1.00	0.76

ADD-S	ADD-S (AUC)	Bowl		Mug		Power Drill		Wood Block		Scissors		Marker	
RKN (Real)		0.17	0.14	0.54	0.52	0.27	0.28	0.05	0.18	0.01	0.16	0.05	0.06
CosyPose (Real)		0.48	0.39	0.81	0.66	0.98	0.79	0.27	0.29	0.22	0.28	0.50	0.44
Self6D++		0.16	0.24	0.03	0.17	0.77	0.62	0.26	0.25	0.11	0.14	0.18	0.17
Ensemble (SSL)		**0.98**	**0.82**	**0.95**	**0.73**	**1.00**	**0.84**	**0.98**	**0.69**	**0.96**	**0.73**	**0.97**	**0.75**
Ensemble (SSL, oc = 1)		0.98	0.82	1.00	0.82	1.00	0.84	1.00	0.74	1.00	0.77	0.99	0.80

ADD-S	ADD-S (AUC)	Large Clamp		Extra Large Clamp		Foam Brick		–		–		–	
RKN (Real)		0.12	0.23	0.05	0.19	0.00	0.20						
CosyPose (Real)		0.86	0.67	**1.00**	**0.81**	0.54	0.46						
Self6D++		0.38	0.32	0.82	0.63	0.19	0.24						
Ensemble (SSL)		**0.97**	**0.78**	0.91	0.73	**0.59**	**0.51**						
Ensemble (SSL, oc = 1)		0.97	0.78	0.96	0.77	0.49	0.43						

[11] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in neural information processing systems*, 2017, pp. 5099–5108.

[12] H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun, "Point transformer," in *Intl. Conf. on Computer Vision (ICCV)*, 2021, pp. 16 239–16 248.

[13] H. Yang, P. Antonante, V. Tzoumas, and L. Carlone, "Graduated non-convexity for robust spatial perception: From non-minimal solvers to global outlier rejection," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 2, pp. 1127–1134, 2020.

[14] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and Model set: Towards common benchmarks for manipulation research," in *Intl. Conf. on Advanced Robotics (ICAR)*, Jul. 2015, pp. 510–517.

[15] G. Wang, F. Manhardt, X. Liu, X. Ji, and F. Tombari, "Occlusion-Aware Self-Supervised Monocular 6D Object Pose Estimation," *IEEE Trans.*

Pattern Anal. Machine Intell., pp. 1–1, 2022.

[16] G. Wang, F. Manhardt, J. Shao, X. Ji, N. Navab, and F. Tombari, "Self6D: Self-supervised monocular 6D object pose estimation," in *European Conf. on Computer Vision (ECCV)*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Nov. 2020, pp. 108–125.

[17] K. Chen, R. Cao, S. James, Y. Li, Y. Liu, P. Abbeel, and Q. Dou, "Sim-to-real 6D object pose estimation via iterative self-training for robotic bin picking," in *European Conf. on Computer Vision (ECCV)*, 2022.

[18] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. Guibas, "Normalized object coordinate space for category-level 6d object pose and size estimation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2642–2651.

[19] S. Zakharov, W. Kehl, A. Bhargava, and A. Gaidon, "Autolabeling 3D Objects with Differentiable Rendering of SDF Shape Priors," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020,

TABLE VI: Evaluation of Ensemble and baselines on 30 TLESS objects.

ADD-S	ADD-S (AUC)	obj_000001		obj_000002		obj_000003		obj_000004		obj_000005		obj_000006	
RKN (Real)		0.71	0.50	0.76	0.53	0.90	0.61	0.19	0.32	0.29	0.40	0.33	0.43
CosyPose (Real)		**0.60**	**0.45**	0.51	0.40	**0.84**	**0.62**	0.41	0.38	**0.89**	**0.68**	**0.94**	**0.70**
Ensemble (SSL)		0.51	0.44	**0.68**	**0.48**	0.69	0.51	0.38	**0.42**	0.36	0.43	0.21	0.36
Ensemble (SSL, oc = 1)		0.33	0.33	0.40	0.33	0.32	0.44	0.85	0.65	0.71	0.49	0.25	0.40

ADD-S	ADD-S (AUC)	obj_000007		obj_000008		obj_000009		obj_000010		obj_000011		obj_000012	
RKN (Real)		0.28	0.40	0.31	0.40	0.33	0.41	0.62	0.52	0.68	0.53	0.74	0.53
CosyPose (Real)		**0.94**	**0.72**	**0.90**	**0.66**	0.90	0.68	0.84	0.63	**0.77**	**0.57**	**0.84**	**0.63**
Ensemble (SSL)		0.25	0.35	0.23	0.33	**0.74**	**0.51**	**0.90**	**0.62**	0.70	0.53	0.72	0.53
Ensemble (SSL, oc = 1)		0.25	0.36	0.25	0.35	0.76	0.52	0.93	0.65	0.72	0.55	0.77	0.55

ADD-S	ADD-S (AUC)	obj_000013		obj_000014		obj_000015		obj_000016		obj_000017		obj_000018	
RKN (Real)		0.46	0.48	0.16	0.26	0.53	0.50	0.75	0.55	0.85	0.62	0.72	0.55
CosyPose (Real)		0.51	0.42	0.71	0.54	**0.79**	**0.58**	**0.89**	**0.64**	**0.94**	**0.72**	**0.93**	**0.76**
Ensemble (SSL)		**0.74**	**0.55**	**0.74**	**0.54**	0.65	0.54	0.74	0.53	0.81	0.59	0.90	0.64
Ensemble (SSL, oc = 1)		0.82	0.59	0.80	0.58	0.68	0.55	0.77	0.55	0.95	0.66	0.98	0.70

ADD-S	ADD-S (AUC)	obj_000019		obj_000020		obj_000021		obj_000022		obj_000023		obj_000024	
RKN (Real)		0.54	0.48	0.11	0.32	0.32	0.40	0.31	0.43	0.32	0.35	0.31	0.39
CosyPose (Real)		**0.79**	**0.59**	**0.73**	**0.55**	**0.75**	**0.61**	**0.70**	**0.55**	**0.87**	**0.67**	0.62	0.50
Ensemble (SSL)		0.40	0.46	0.13	0.30	0.41	0.41	0.52	0.48	0.66	0.52	**0.66**	**0.52**
Ensemble (SSL, oc = 1)		0.40	0.44	0.26	0.33	0.38	0.39	0.61	0.52	0.75	0.58	0.98	0.69

ADD-S	ADD-S (AUC)	obj_000025		obj_000026		obj_000027		obj_000028		obj_000029		obj_000030	
RKN (Real)		0.40	0.46	0.40	0.47	0.32	0.40	0.42	0.42	0.14	0.34	0.79	0.58
CosyPose (Real)		**0.86**	**0.65**	**0.83**	**0.64**	**0.91**	**0.69**	**0.85**	**0.69**	**0.94**	**0.72**	0.93	0.71
Ensemble (SSL)		0.63	0.55	0.82	0.61	0.79	0.57	0.65	0.46	0.58	0.49	**0.97**	**0.73**
Ensemble (SSL, oc = 1)		0.82	0.60	0.90	0.64	0.85	0.60	0.72	0.51	0.76	0.59	1.00	0.75

pp. 12 224–12 233.

[20] K. Zhang, Y. Fu, S. Borse, H. Cai, F. Porikli, and X. Wang, "Self-supervised Geometric Correspondence for Category-level 6D Object Pose Estimation in the Wild," in *Intl. Conf. on Learning Representations (ICLR)*, Feb. 2023.

[21] X. Deng, Y. Xiang, A. Mousavian, C. Eppner, T. Bretl, and D. Fox, "Self-supervised 6D Object Pose Estimation for Robot Manipulation," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2020, pp. 3665–3671.

[22] X. Li, Y. Weng, L. Yi, L. Guibas, A. L. Abbott, S. Song, and H. Wang, "Leveraging SE(3) Equivariance for Self-Supervised Category-Level Object Pose Estimation," in *Advances in Neural Information Processing Systems (NIPS)*, Dec. 2021.

[23] W. Sun, A. Tagliasacchi, B. Deng, S. Sabour, S. Yazdani, G. E. Hinton, and K. M. Yi, "Canonical Capsules: Self-Supervised Capsules in Canonical Pose," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 34, 2021, pp. 24 993–25 005.

[24] X. Wang, Y. Jin, Y. Cen, T. Wang, B. Tang, and Y. Li, "LighTN: Light-weight Transformer Network for Performance-overhead Tradeoff in Point Cloud Downsampling," *arXiv*, no. arXiv:2202.06263, Feb. 2022.

[25] Y. Lin, L. Chen, H. Huang, C. Ma, X. Han, and S. Cui, "Task-Aware Sampling Layer for Point-Wise Analysis," *IEEE Trans. Vis. Comput. Graph.*, pp. 1–1, May 2022.

[26] Y. Qian, J. Hou, Q. Zhang, Y. Zeng, S. Kwong, and Y. He, "MOPS-Net: A Matrix Optimization-driven Network forTask-Oriented 3D Point Cloud Downsampling," *arXiv*, no. arXiv:2005.00383, Apr. 2021.

[27] X. Yan, C. Zheng, Z. Li, S. Wang, and S. Cui, "PointASNL: Robust Point Clouds Processing Using Nonlocal Neural Networks With Adaptive Sampling," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, pp. 5588–5597.

[28] E. Nezhadarya, E. Taghavi, R. Razani, B. Liu, and J. Luo, "Adaptive Hierarchical Down-Sampling for Point Cloud Classification," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, pp. 12 953–12 961.

[29] I. Lang, A. Manor, and S. Avidan, "SampleNet: Differentiable Point Cloud Sampling," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020, pp. 7575–7585.

[30] J. Yang, Q. Zhang, B. Ni, L. Li, J. Liu, M. Zhou, and Q. Tian, "Modeling Point Clouds With Self-Attention and Gumbel Subset Sampling," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019, pp. 3318–3327.

[31] O. Dovrat, I. Lang, and S. Avidan, "Learning to Sample," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019, pp. 2760–2769.

[32] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *Intl. Conf. on Machine Learning (ICML)*. JMLR. org, 2017, pp. 136–145.

[33] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," in *Advances in Neural Information Processing Systems (NIPS)*, 2019, pp. 9558–9570.

[34] S. Gould, R. Hartley, and D. Campbell, "Deep Declarative Networks," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 44, no. 8, pp. 3988–4004, Aug. 2022.

[35] P. Donti, B. Amos, and J. Z. Kolter, "Task-based End-to-end Model Learning in Stochastic Optimization," in *Advances in Neural Information Processing Systems*, vol. 30, Nov. 2017.

[36] M. V. Pogančić, A. Paulus, V. Musil, G. Martius, and M. Rolinek, "Differentiation of Blackbox Combinatorial Solvers," in *Intl. Conf. on Learning Representations (ICLR)*, Mar. 2020.

[37] A. Paulus, M. Rolinek, V. Musil, B. Amos, and G. Martius, "CombOpt-Net: Fit the Right NP-Hard Problem by Learning Integer Programming Constraints," in *Intl. Conf. on Machine Learning (ICML)*, Jul. 2021, pp. 8443–8453.

[38] P.-W. Wang, P. Donti, B. Wilder, and Z. Kolter, "SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver," in *Proceedings of the 36th International Conference on Machine Learning*, May 2019, pp. 6545–6554.

[39] C. Wang, D. Gao, K. Xu, J. Geng, Y. Hu, Y. Qiu, B. Li, F. Yang, B. Moon, A. Pandey, A. Aryan, J. Xu, T. Wu, H. He, D. Huang, Z. Ren, S. Zhao, T. Fu, P. Anthireddy, W. Wang, J. Shi, R. Talak, H. Wang,

H. Yu, S. Wang, A. Kashyap, R. Bandaru, K. Dantu, J. Wu, L. Carlone, M. Hutter, and S. Scherer, "PyPose: A library for robot learning with physics-based optimization," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, arXiv preprint: 2209.15428 (pdf).

[40] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Q. Chen, J. Ortiz, D. DeTone, A. S. Wang, S. Anderson, J. Dong, B. Amos, and M. Mukadam, "Theseus: A Library for Differentiable Nonlinear Optimization," in *Advances in Neural Information Processing Systems (NIPS)*, Oct. 2022.

[41] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Intl. Conf. on Computer Vision (ICCV)*, 2017, pp. 2980–2988.

[42] J. Shi, H. Yang, and L. Carlone, "Optimal pose and shape estimation for category-level 3D object perception," in *Robotics: Science and Systems (RSS)*, 2021.

[43] G. Pavlakos, X. Zhou, A. Chan, K. Derpanis, and K. Daniilidis, "6-dof object pose from semantic keypoints," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2017.

[44] K. Schmeckpeper, P. Osteen, Y. Wang, G. Pavlakos, K. Chaney, W. Jordan, X. Zhou, K. Derpanis, and K. Daniilidis, "Semantic keypoint-based pose estimation from single rgb frames," *arXiv preprint arXiv:2204.05864*, 2022.

[45] K. M. Tavish and T. D. Barfoot, "At all costs: A comparison of robust cost functions for camera correspondence outliers," in *Conf. Computer and Robot Vision*. IEEE, 2015, pp. 62–69.

[46] H. Yang, J. Shi, and L. Carlone, "TEASER: Fast and Certifiable Point Cloud Registration," *IEEE Trans. Robotics*, vol. 37, no. 2, pp. 314–333, 2020.

[47] B. k P Horn and E. J. Weldon, "Direct methods for recovering motion," *Intl. J. of Computer Vision*, vol. 1, no. 2, pp. 51–76, 1988.

[48] K. Arun, T. Huang, and S. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 9, no. 5, pp. 698–700, sept. 1987.

[49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NIPS)*, Dec. 2019, pp. 8024–8035.

[50] C. Wang, D. Xu, Y. Zhu, R. Martin-Martin, C. Lu, L. Fei-Fei, and S. Savarese, "Densefusion: 6d object pose estimation by iterative dense fusion," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019, pp. 3338–3347.

[51] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes," in *Robotics: Science and Systems (RSS)*, 2018.

[52] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "DeepIM: Deep Iterative Matching for 6D Pose Estimation," in *European Conf. on Computer Vision (ECCV)*, 2018, pp. 683–698.