Pose Estimation with Dense Descriptors and Superpixel-Enhanced Instance Segmentation

Wei Gao
MIT EECS
Cambridge, MA
weigao@mit.edu

Rishi Sundaresan
MIT EECS
Cambridge, MA
rishisun@mit.edu

Ethan Weber
MIT EECS
Cambridge, MA
ejweber@mit.edu

***Abstract*—In this paper, we present an approach to pose estimation for a known class of objects. We optimize and combine computer vision algorithms including Mask R-CNN and Simple Linear Iterative Clustering (SLIC) for instance segmentation to create inputs for a novel pose estimation iterative algorithm that allow it to locate all objects in a given scene. We show that this pipeline performs well in cluttered scenes, where–without instance segmentation–applying the iterative pose estimation algorithm fails. This has applications in robot planning, control, and locomotion, where in the case of robotic manipulation we wish to locate objects and keep track of the visual features that describe the objects.**

***Index Terms*—pose estimation, dense descriptors, instance segmentation, superpixels, robotic manipulation**

I. Introduction

Pose estimation, or localizing objects in scenes, is vital to robot manipulation tasks. In order to manipulate objects in the world, it's important to have a perception system to estimate the state of the world (and the objects in it). In this project, we make create a pipeline for pose estimation of multiple objects.

We will build on work from the Dense Object Nets paper [1], which transform RGB images into their corresponding dense descriptor images. Given an object in descriptor space, we use an iterative algorithm to find the pose of the object in the scene.

However, plainly this method fails when multiple objects are introduced into a cluttered scene, as the method is unable to accurately estimate the pose of each of multiple overlapping objects. We solve this problem by first cropping the dense descriptor images with instance segmentation and superpixel refinement. On the cropped objects in descriptor space, we then use the iterative algorithm to find the pose of every object in the scene.

II. Proposed Pipeline

We propose a novel pipeline for pose estimation from images. First, we perform instance segmentation using Mask R-CNN trained on synthetic images (formed with parts of real images), which outputs a mask for each instance in a given image. Next, for each instance mask, we use a Simple Linear Iterative Clustering (SLIC) algorithm [2] with optimized hyperparameters to enhance the mask. Finally, we use the refined masks along with dense descriptors for pose estimation of each instance. The general structure of our pipeline is shown in Figure 1. The dense descriptors used are from the work Dense Object Nets [1], which is described in Figure 4. The data we use from Dense Objects Nets is as follows:

- RGB images
- depth images
- background subtracted binary masks

Note that the pose estimation makes use of the depth images to obtain the 3d point cloud for pose estimation. Figure 5 clearly shows the data we have access to and use for our proposed pipeline.

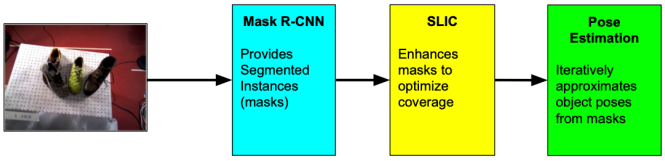

Fig. 1. General Structure of Pipeline. We use Mask R-CNN from [3] for instance segmentation and base our SLIC (Simple Iterative Linear Clustering) superpixel segmentation algorithm from [2].

In the rest of this section, we describe the steps of our pipeline in detail.

A. Instance Segmentation

For the instance segmentation network, we use Facebook's Detectron [4] implementation of Mask R-CNN (regional convolutional neural network), which uses Caffe2 [5]. Mask R-CNN takes as input an RGB image and regresses the following 3 items: bounding boxes, instance masks, and classifications. This is done in a two-stage approach. First, proposals are generated for which regions of the image are likely to contain objects and second, bounding boxes, masks, and classes are predicted for the proposals. Mask R-CNN is an extension of Faster R-CNN [6], where Mask R-CNN adds an instance segmentation branch and also introduces a new technique called RoIAlign. RoIAlign improves upon RoI-pooling by using bilinear interpolation, which is described in the Mask R-CNN paper [3]. For the purpose of our work, we are most interested in the Mask R-CNN branch that predicts the mask for all object instances. Furthermore, we operate on only one class of objects for this work, so the class prediction is not

very relevant to us. We show a diagram from the original Mask R-CNN paper depicting its input and output in Figure 2.

We use transfer learning by starting with weights that were pretrained on the COCO dataset. We fine-tune the weights to our custom dataset, which we explain in a later section. The output of the Mask R-CNN module are masks that can be post-processed by a SLIC algorithm.

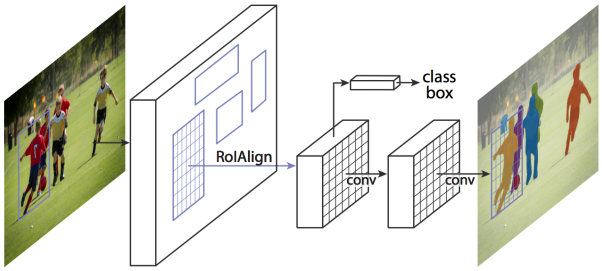

Fig. 2. This is the framework for Mask R-CNN [3].

B. SLIC Superpixel Refinement

After running an RGB image through the trained Mask R-CNN network, we take each object instance (predicted mask) and refine it based on the SLIC superpixel algorithm [2]. This is implemented to slightly enlarge the mask to increase the number of ground-truth mask pixels contained in the predicted mask. We believe this is helpful because Mask R-CNN is trained on synthetic images with possibly imperfect masks– given that our data comes from the Dense Object Nets dataset (without human labels).

SLIC is a k-means clustering algorithm that segments the image into k portions ("superpixels"), with centers sampled from pixels at regular intervals. The clustering algorithm is based on a distance measure that incorporates spatial proximity and color proximity, meaning that crossing a border of two colors contributes to a large distance.

We use SLIC as follows. From instance segmentation, for each object, we receive the rgb image of the object bounding box and the corresponding predicted mask. We run the SLIC algorithm on the rgb image to generate superpixels, and then enhance the predicted mask using the process below. Note that the predicted mask is an 1-channel image with pixel value > 0 if the pixel location is in the mask, and pixel value $= 0$ if the pixel location is not in the mask.

---

foreach *superpixel in SLIC segmentation* **do**
  if *superpixel contains pixel locations in mask* **then**
    foreach *pixel location in superpixel not in mask* **do**
      add pixel location to mask;
    end
  end
end

Algorithm 1: Superpixel Enhancement to Mask

The algorithm will essentially "fill in" areas where the mask does not cover an entire superpixel in the corresponding rgb image. An example of this process is shown in Figure 6.

After augmenting the predicted mask using this refinement, we feed the mask into the third stage of our pipeline. The main hyperparameter of the SLIC algorithm is k, the number of clusters (superpixels) generated by the segmentation. We call this hyperparameter the *resolution* of the SLIC algorithm, and we describe how we optimize it in section IIIB.

C. Pose Estimation

On a high-level, our pose estimation algorithm works as follows. Through depth images of our dataset, we are able to learn a geometric template of the object class in question (template is shown in Figure 3). We then color each point in the template based on a "learned feature" of what part of the object (in this case shoe) the point corresponds to. For example, as shown in Figure 3, the laces area of the shoe is colored green, while the heel area is purple. For a given RGBD image, we use the model from Dense Object Net to create learned feature outlines of shoes in the image along with depth (as shown in figure 4), and then we iteratively "fit" the template to the learned feature outlines and depths in the image. We will now explain this process more in detail.

To express the pose of objects, we created object models using 3d reconstruction algorithm ElasticFusion [7]. (The input to this algorithm was many depth images taken from the dataset of the particular scene we are interested in. This comes from a video stream of RGB-D images taken by the robotic arm camera moving around the static scene.) The ElasticFusion system outputs de-noised geometric template of the objects. After 3d reconstruction, we colored each vertex in the geometric template with Dense Object Net [1] features. We use the camera pose from the ElasticFusion system to project a vertex in 3d space into image pixels. In this way, given an input RGB-D and descriptor image, each vertex has a corresponded learned feature if the vertex is visible. We iterate all the input images and use the averaged descriptor value for each vertex in the geometric template. An example geometric template is shown in Fig. 3.

With the geometric template colored by the learned feature, we can estimate the pose given an observation using the correspondence established by the learned feature. One solution is to use the "feature ICP" [8]: for each vertex in the geometric template, search its nearest neighbour from the observation, in the feature space. After the nearest neighbour is ready, the pose is estimated by minimizing the following objective function:

$$\min_T \Sigma_{i \in \text{template}} |T v_i - u_i|^2 \quad (1)$$

where u_i is the 3d position of the nearest neighbour of the ith vertex. This optimization is a least squares problem and we use the Gauss-Newton algorithm to solve it.

This method works for many setups. However, the nearest neighbour correspondence is not robust to noise, outliers and

occlusions. To resolve it, we replace the nearest neighbour target u_i with multiple weighted targets.

$$u_{\text{multilink}-i} = \Sigma_{j \in \text{observation}} w_{ij} u_j \tag{2}$$

where the weight $w_i j$ is

$$w_{ij} = \text{Gaussian}(f_i; f_j, \Sigma) \tag{3}$$

is the guassian weight. f_i and f_j are the learned feature value. The computation of Equ. 2 is expensive by brute-force, an efficient method is presented in [9].

Fig. 3. The geometric template used for the pose estimation of shoes. From left to right: the RGB image (visualization only); the point cloud of the shoe model; the learned feature of each vertex.

III. Training and Optimization

In this section, we show how we trained and optimized our models for the first two steps of the pipeline.

A. Instance Segmentation Training

In this first step of our pipeline, we train and use a Mask R-CNN model for instance segmentation. We train Mask R-CNN by using the data provided from Dense Object Nets. In Figure 5, we see an example of a background subtracted binary mask from the dataset. This mask was created by subtracting the background plane from the image (based on the depth image and 3d reconstruction of the scene). This works great for single shoe images, but when there are multiple shoes, such as in the RGB image in Figure 4, the background subtracted mask will not be informative. Rather, there is no way to identify the shoes as instances, and the shoes will clump together if overlapping. To solve this data limitation problem, we create an augmented dataset and train Mask R-CNN for instance segmentation.

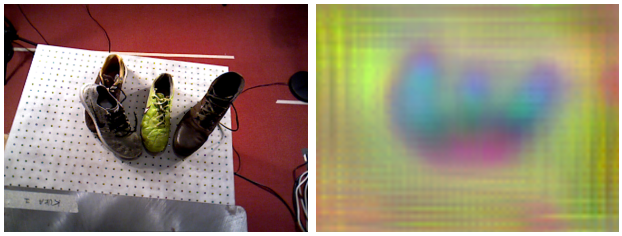

Fig. 4. Here we show the output of Dense Object Nets [1]. In this case, the network was trained to identify correspondences among shoes. Using network weights provided by Dense Objects Nets, we run inference for the RGB image. On right we see the output dense descriptor image. Visualized as color space, we can see that the same colors represent the same part of the shoes–even across different classes.

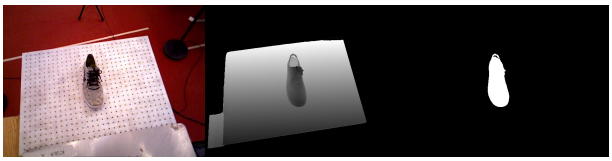

Fig. 5. Here we show the data provided from the Dense Object Nets [1] work. We have RGB images (left) with corresponding depth images (middle) and background subtracted binary masks (right). Note that this dataset also has information such as camera pose for multiple images of the same scene, but we do not need that data for our research.

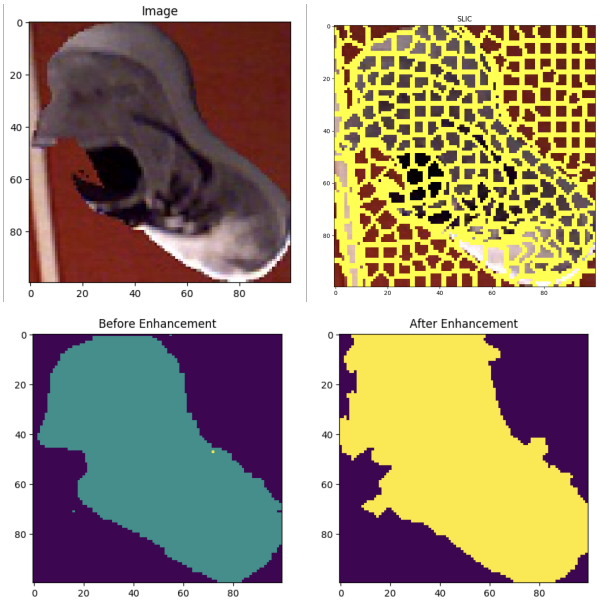

Fig. 6. Here we show an example of enhancing a mask through SLIC segmentation, with a resolution (number of clusters) of 324. SLIC is applied to the original image (top left), creating a segmented image (top right). The mask (before enhancement) is then refined to create a final mask (after enhancement). In this example, the refined mask contains extra pixels but also covers more pixels of the image than the original mask (see near the opening of the shoe.

1) Data Synthesis into COCO Format: Because we have many RGB images of single shoes with corresponding binary masks, we can create a new dataset of **multiple** synthetic shoes. We do this by randomly choosing 1-7 single shoe images from the Dense Object Nets dataset. We then choose

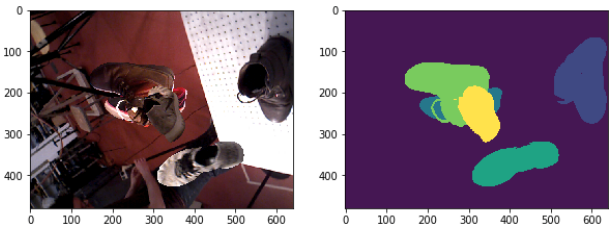

Fig. 7. On the left we show a synthetic RGB image and on the right we show the corresponding masks. We've written code to store all the annotations in COCO format. This allows for good integration with the Mask R-CNN implementation we use.

one of the images to be the background. Finally, we then take all the remaining images and paste only the masked RGB pixels (of the shoe) and replace the pixels of the "background" to be these shoes. We do this in sequential order and keep track of the new masks that result from the merging process. Here we show the psuedo code for the merging process and illustrate the results in Figure 7.

randomly choose 1-7 RGB images and binary masks;
use the first RGB image as the background;
for *current RGB image and masks in remaining* **do**
  replace first RGB image pixels with current RGB image pixels;
  update first mask according to keep track of which pixels belong to which shoe;
end

Algorithm 2: Synthetic Image Merging

2) Mask R-CNN Training: After completing the code to merge images and store the annotations, we create a training and validation dataset with images like those shown in Figure 7. Our generated training dataset has 10,000 synthetic images and our validation dataset has 2,000 images. Recall that each image has a random choice of 1 to 7 shoes per image. Also, we focus on shoes because we only care about robotic manipulation and pose estimation for a single class type in our work.

For our Mask R-CNN configuration with Detectron, we use a learning rate of 0.0025 with a weight decay of 0.0001. We use a γ value of 0.1, which drops the learning rate by a factor of 10 at iterations 0, 30000, and 40000. We terminate training at 60000 iterations. For Mask R-CNN specifically, we have an output mask resolution of 28, but note that the values are not binary and encode enough information to expand to a high resolution. We use RoIAlign, a dilation of 1, and an RoI sampling ratio of 2.

During training, we initialize our network with pretrained weights on ImageNet. This form of transfer learning is extremely helpful because many low-level image features can be used and fine-turned for our shoes. We use a batch size of 256 and take the top 2000 values after non-maximal suppression.

Bounding Box Evaluation Metrics on Validation Set

Metric	IoU	Area	MaxDets	Value
AP	0.50:0.95	all	100	0.844
AP	0.50	all	100	0.977
AP	0.75	all	100	0.916
AP	0.50:0.95	small	100	0.256
AP	0.50:0.95	medium	100	0.726
AP	0.50:0.95	large	100	0.901
AR	0.50:0.95	all	1	0.239
AR	0.50:0.95	all	10	0.864
AR	0.50:0.95	all	100	0.866
AR	0.50:0.95	small	100	0.307
AR	0.50:0.95	medium	100	0.733
AR	0.50:0.95	large	100	0.925

TABLE I

Segmentation Evaluation Metrics on Validation Set

Metric	IoU	Area	MaxDets	Value
AP	0.50:0.95	all	100	0.846
AP	0.50	all	100	0.966
AP	0.75	all	100	0.907
AP	0.50:0.95	small	100	0.145
AP	0.50:0.95	medium	100	0.678
AP	0.50:0.95	large	100	0.909
AR	0.50:0.95	all	1	0.239
AR	0.50:0.95	all	10	0.867
AR	0.50:0.95	all	100	0.868
AR	0.50:0.95	small	100	0.227
AR	0.50:0.95	medium	100	0.720
AR	0.50:0.95	large	100	0.929

TABLE II

There were some other configuration parameters used in the network, but we only highlight these because Mask R-CNN is very well known in the field of instance segmentation.

For the validation set of 2,000 images, we acknowledge that the images are synthetic while the performance we actually care about is on real images of multiple shoes. Because we do not have the ground truth data for real images of multiple shoes and their corresponding masks, we rely on this test set to assess statistical values (which are shown in the experimental results section). However, we have empirically tested on real images of multiple shoes with good results. We show an example in Figure 10.

3) Mask R-CNN Results: We show the average precision (AP), average recall (AR), and other metrics in Table III-A3 for bounding box and Table III-A3 for segmentation on the validation data set. The results are notably good with high precision and recall in almost all cases. The statistics shown in the chart are explained in the COCO detection evaluation documentation [10].

B. SLIC Superpixel Resolution Optimization

We optimize the *resolution* of our SLIC algorithm. To accomplish this, we first needed a dataset with ground truth and predicted masks for each detected instance.

1) Generated Dataset: We create a SLIC dataset from the 2,000 test images by running inference with the trained Mask R-CNN network. Inference for a given RGB image in the

test set outputs the bounding boxes and masks of detected objects (shoes in our case). We iterate through each image and detected bounding box to create a dataset with many entries of the following form:

- cropped RGB image
- cropped ground truth annotation
- cropped predicted mask

We create the **cropped RGB image** by taking the detected bounding box, the **cropped ground truth annotation** by cropping the ground truth mask label image, and the **cropped predicted mask** by taking only the predicted mask within the bounding box. Each cropped image is resized to 100×100 pixels. We illustrate this procedure in Figure 8.

2) Optimizing the SLIC Resolution: We now optimize the SLIC resolution on the dataset. To do so, we must first define a loss function for a predicted mask and a ground truth mask. There are three types of pixels in our mask:

1) *correct pixels*: pixels that are in the predicted mask and in the ground truth mask
2) *missing pixels*: pixels that are not in the predicted mask but in the ground truth mask
3) *extra pixels*: pixels that are in the predicted mask but not in the ground truth mask

Typical loss functions such as intersection over union are not applicable to our setting, as for our application, it is much worse to miss pixels than to have extra pixels. This observation stems from the third stage of our pipeline, where we believe pose estimation can be done accurately with extra pixels, but its accuracy would be worsened with an incomplete mask. Thus, we define a modified loss function with inputs of missing pixels m, extra pixels e, and total number of pixels in the ground-truth mask n:

$$L(m, e, n) = \frac{1}{n}(25m + e)$$

Note that the $\frac{1}{n}$ term ensures that bigger masks are not unfairly given larger losses. We determine the weighting factor of 25 based on our intuition of the effect on pose estimation and the empirical ability of convergence to an optimal resolution. More specifically, we find that this weighting factor allows for convergence to a non-trivial solution, while other weighting factors resulted in unfeasible extreme optimal solutions.

Next, we find the optimal resolution that minimizes this loss for the dataset. To accomplish this, we first set a wide range of possible resolution values, in our case $[0, 1000]$. (Any resolution over 1000 would have on average 10 pixels per superpixel, which is extremely overfitting to specific pixels and would not result in a significant solution). We then sampled 30 resolutions at regular intervals from this range and calculated their corresponding average losses. For any given resolution, one can compute the average loss as below.

The average losses of each of the 30 resolutions are shown in Figure 9.

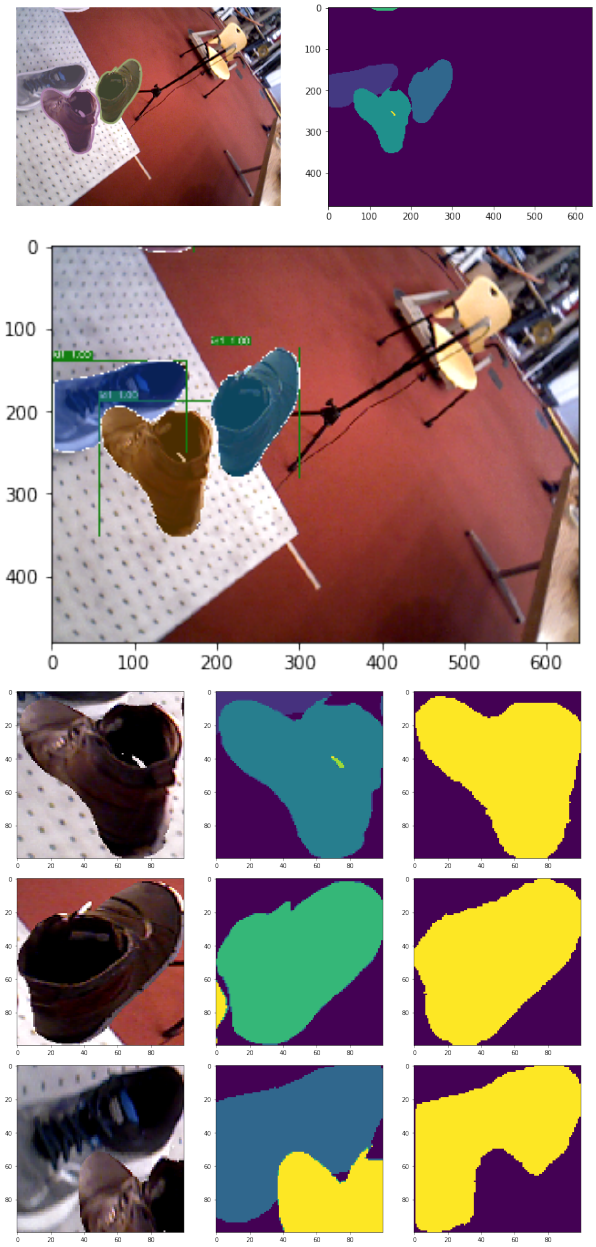

Fig. 8. Here we show how we create the dataset for the SLIC optimal parameter searching. The top 2 images show the ground truth segmentation masks (overlayed on the RGB image) and separately. In the second (large) image, we show the result of running inference with the trained Mask R-CNN network. To create data for SLIC optimization, we create images of the format of the last 3 rows. The first column is the cropped RGB based on the predicted bounding box. The middle column is a cropped portion of the ground truth mask. The right column is the predicted mask from Mask R-CNN. Note that we have resized all bounding boxes of instances to 100×100 pixels.

Based on these results, we narrow our search space of resolutions to [250,550]. In this range, we perform a modified version of stochastic gradient descent (SGD). We initialize two resolutions, at 330 and 333. We then perform the SGD update equation to the resolution k using a learning rate η.

$$k_{t+1} = k_t - \eta \frac{\partial avgLoss}{\partial k_t}$$

Pick 500 random data points (image, ground-truth mask, predicted mask) from generated dataset
total-loss = []
foreach *(image, ground-truth-mask, predicted-mask)* **do**
    new-mask = enhance-mask(predicted-mask, resolution)
    m = missingPixels(new-mask,ground-truth-mask)
    e = extraPixels(new-mask,ground-truth-mask)
    n = size of ground-truth-mask
    append L(m,e,n) to total-loss
end
return mean(total-loss);

Algorithm 3: Calculating Average loss for Resolution

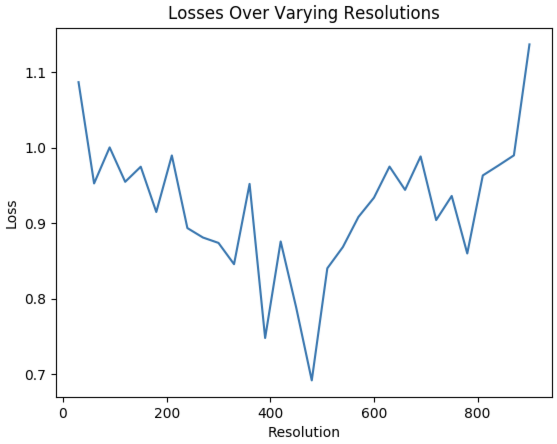

Fig. 9. Here we show average losses over varying resolutions from 0 to 1000. As shown by the graph, we can narrow our search space of the optimal resolution that minimizes loss to the resolution range of [250,550]

Since it is infeasible to compute $\frac{\partial avgLoss}{\partial k_t}$ exactly, we approximate it using the slope of the secant line between k_t and k_{t-1}:

$$\frac{\partial avgLoss}{\partial k_t} \approx \frac{avgLoss(k_t) - avgLoss(k_{t-1})}{k_t - k_{t-1}}$$

We perform this version of SGD for a maximum amount of iterations, with a decaying learning rate, and then we use another key modification to the algorithm. Due to the possibly high variance of results with respect to resolution k (since segmentations are very dependent on image characterstics), and the possibility of the loss function not being convex with respect to k, rather than outputing the final k after SGD, we store all $AvgLoss(k_i)$ for all i. We then return the optimal i as:

$$i^* = \arg\min_i AvgLoss(k_i)$$

The optimal resolution we calculate is k_{i^*}. When we performed this procedure, we arrived at an optimal resolution of $k = 324$. We used this optimal hyperparameter for the SLIC step of our pipeline. With this resolution, we achieve an average loss of 0.59. Using the same loss function, without SLIC enhancement, the average loss of our masks is 1.21. Thus, the SLIC enhancement decreases our average loss by about half.

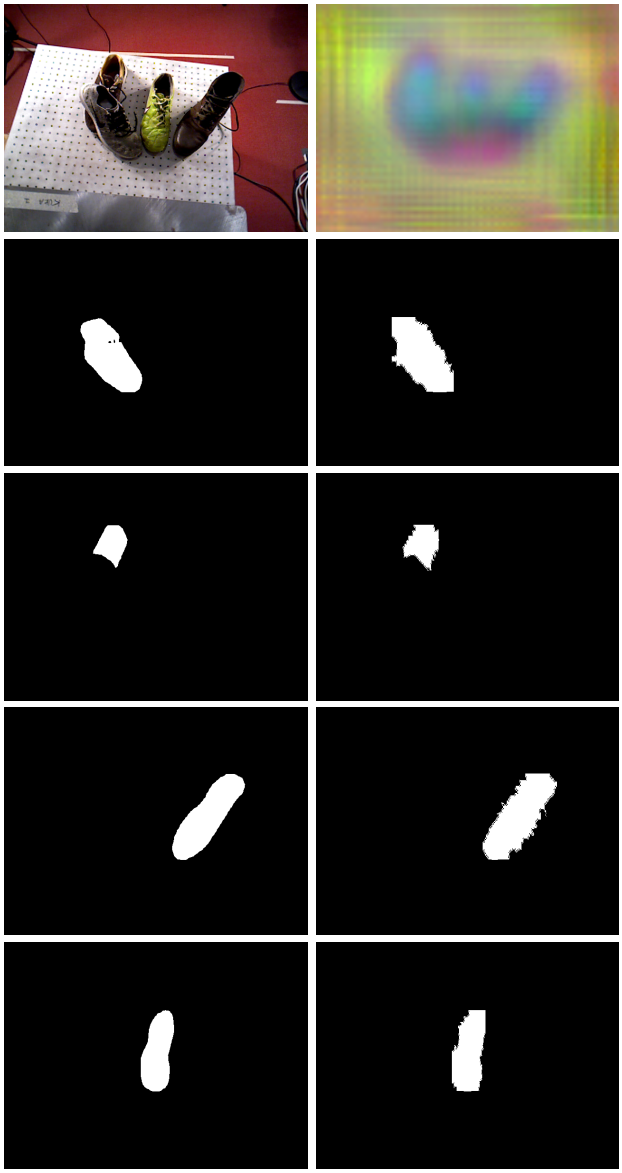

Fig. 10. Here we show the input needed for dense descriptor pose estimation. The depth image is not shown, but that is very vital. The RGB image is technically not needed for the pose estimation. The left masks show the output from Mask R-CNN and the right images show the masks after SLIC refinement. We experiment with both masks and pass these to the pose estimator.

IV. Experimental Results

Here we show experimental results for our proposed perception and pose estimation pipeline on shoe data.

A. Mask R-CNN + SLIC Output

After instance segmentation and SLIC refinement, we resize each of the 100×100 pixel masks back to the original size

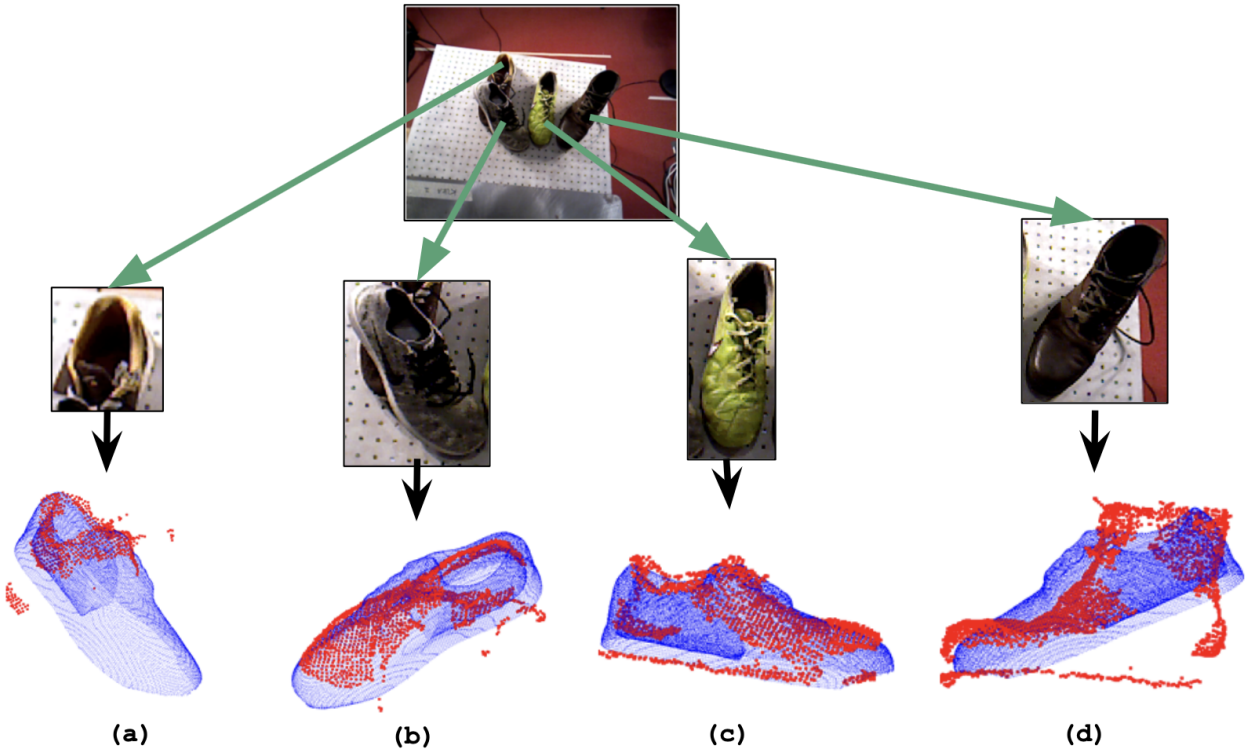

Fig. 11. The pose estimation result using the input from Fig. 10. Our method can handle the variation on the appearance, size and shape within the shoe category. Furthermore, the proposed method is robust to occlusion and yields decent pose estimation under severe partial view, as shown in (a). Thus, it performs well on cluttered scenes

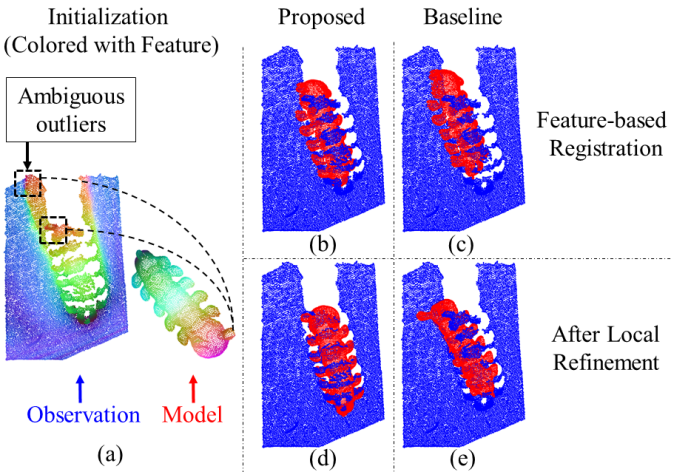

Fig. 12. A feature-based global registration under ambiguous outliers and strong occlusion. (a) shows the initialization o the registration colored by the learned feature. (b) and (c) are the feature-based registration by our method and the baseline. (d) and (e) show the final alignment using 3d local refinement initialized from (b) and (c). The proposed method converges to the correct pose while the baseline method is trapped to bad alignment.

and pass the data to the pose estimator stage. This is shown in Figure 10. At this point, we have the following data which is used by the pose estimator:

- RGB image
- dense descriptor image
- depth image
- list of raw instance segmentation masks
- list of SLIC refined instance segmentation masks

B. *Pose Estimation Results*

Overall, our novel pose estimation pipeline performs well on cluttered images and images with multiple shoes. An example of this performance is shown in Fig. 11. Points in red are from the depth image segmented by the instance segmentation masks, while points in blue are the geometric template. The proposed method is able to perform pose estimation on instances with variations on the appearance, size and shape of the shoes. The proposed method is also very robust to occlusions in the input, as shown in Fig. 11 (a).

It is difficult to evaluate the model's performance due to a lack of ground truth data. However, we qualitatively judge the model's performance on a variety of images based on the fit to the depth image point cloud, and we find that the model performs well over our dataset.

To further test the robustness of our method, we compare it with a modified feature ICP algorithm [11]. In addition, to show that our method can generalize to other classes of objects, we train and evaluate the pose estimation model on caterpillars rather than shoes. On a high level, our pose

	success rate	time [ms]
Proposed	29/30	13
Feature ICP	25/30	34

TABLE III
THE SUCCESS RATE AND SPEED ON THE FEATURE-BASED GLOBAL REGISTRATION.

estimation method only requires a model and point cloud to fit to.

Fig. 12 shows an example registration. Note that we treat the background as outliers. As shown in Fig. 12 (a), the observation (RGBD cloud) is under severe occlusion and contains very ambiguous outliers. Fig. 12 (b) and (c) show feature-based registration by our method and the "feature" ICP. The proposed method is more robust to the outliers and occlusion. Fig. 12 (d) and (e) show the final alignment using local refinement initialized from (b) and (c). The proposed method converges to correct pose while the baseline is trapped to bad alignment. Table III summaries the success rate of both methods on 30 RGBD images with different view points and lighting conditions. Our method has a higher success rate and is more efficient than the baseline method.

V. CONCLUSION

In summary, we propose a novel pose estimation pipeline to locate multiple possibly overlapping objects in scenes. We utilize Mask R-CNN to segment object instances (represented as mask outlines) from an image, utilize a superpixel segmentation algorithm (SLIC) to refine the masks, and then feed the masks into the pose estimation algorithm for successful object localization. We show that our pipeline performs well for the object class of shoes. We also show that our method is robust and can be applied to other object classes.

VI. FUTURE WORK

In terms of future work, there are a few areas which we hope to address and improve upon. In particular, we hope to test the pose estimation with more numerical metrics. Currently we were limited to empirical inspection of results (because we don't actually have ground truth pose), but we may be able to circumvent this problem by testing with rendered synthetic data (such as from a game engine).

Further, we also noticed that the SLIC step did not produce notably different results than the direct output masks from Mask R-CNN. We suspect this may be because Mask R-CNN is quite successful. Theoretically, we maintain our intuition that the SLIC step can be beneficial, but we hope to test results numerically after setting up a synthetic test environment (explained briefly above).

In terms of practical use, our next step is to fully implement this system on a robot to perform robotic manipulation for multiple objects of a known class. This work is novel and directly integrates with work from Dense Object Nets [1].

VII. DIVISION OF LABOR

Ethan Weber (instance segmentation): I worked on instance segmentation for this project which involved data processing, training, and making sure that our code integrated well with the Dense Object Nets [1] code.

Rishi Sundaresan (SLIC refinement): I worked on implementing the SLIC refinement part of the pipeline, as well as using computational methods to optimize SLIC parameters.

Wei Gao (pose estimation): I worked on the pose estimation part of this project. In particular, I propose a novel filter-based pose estimation algorithm [9] and applied it to our problem.

REFERENCES

[1] P. R. Florence, L. Manuelli, and R. Tedrake, "Dense object nets: Learning dense visual object descriptors by and for robotic manipulation," *CoRR*, vol. abs/1806.08756, 2018.

[2] K. S. A. L. P. F. Radhakrishna Achanta, Appu Shaji and S. Susstrunk, "Slic superpixels." http://www.kev-smith.com/papers/SLIC_Superpixels.pdf.

[3] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[4] "Detectron." https://github.com/facebookresearch/Detectron.

[5] "Caffe2." https://caffe2.ai/.

[6] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[7] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "Elasticfusion: Dense slam without a pose graph,"

[8] M. Korn, M. Holzkothen, and J. Pauli, "Color supported generalized-icp," in *Computer Vision Theory and Applications (VISAPP), 2014 International Conference on*, vol. 3, pp. 592–599, IEEE, 2014.

[9] W. Gao and R. Tedrake, "Filterreg: Robust and efficient probabilistic point-set registration using gaussian filter and twist parameterization,"

[10] "Detection evaluation." http://cocodataset.org/detection-eval.

[11] G. C. Sharp, S. W. Lee, and D. K. Wehe, "Icp registration using invariant features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 90–102, 2002.

[12] "Coco data format." http://cocodataset.org/format-data.