SparKey: Sparse Keypoint Discovery for Robotic Manipulation

Ethan Weber
MIT EECS
ejweber@mit.edu

Abstract

Robots are becoming increasingly common in the manufacturing, medical, food, and automotive industries. For each of these fields, the task of robotic manipulation–being able to directly control an object along a trajectory via "pick and place"–is extremely important. In order to accomplish a "pick and place" task, both *perception* and *planning and control* pipelines are needed. In our work, we present a novel perception pipeline called **SparKey** (short for "Sparse Keypoints") for sparse keypoint discovery. Improving upon previous work, our pipeline consists of instance segmentation, optimal keypoint detection for 3D pose estimation, and tracking through occlusions. Our main contribution is our pipeline for finding the best keypoints of objects with self-supervised methods. Furthermore, our sparse keypoint representation is ideal input for *planning and control* pipelines because it's most computationally efficient for planning a trajectory with a small number of points. Additionally, this representation has keypoints for optimal pose estimation, as demonstrated in related work (1). We motivate all our work with shoe data, but our methods can be applied to any object we'd like a robot to manipulate.

Keywords: Keypoint Detection, Self-Supervised, Computer Vision, Robotic Manipulation

1 Introduction

How should robots perceive the world? Robots are becoming integrated into fields such as manufacturing, agriculture, medicine, food preparation, and more. In almost all of these fields, the robots are physically interacting with the real world through robot manipulation. To be successful at these tasks, the robots must have a visual understanding of the world.

In this paper, we focus on visual understanding for robotic manipulation. Typical robot manipulation algorithms include two major pipelines: (1) perception, and (2) planning and control. Perception is used to represent the state of the world, and planning and control is used to change it in a desirable way. Trajectory optimization is one popular algorithm for planning and control, which requires knowledge about the state of the world coming from the perception pipeline. In this research, we present a new perception pipeline called SparKey: Sparse Keypoint Discovery for Robotic Manipulation. SparKey is better than existing perception pipelines because (1) it works for multiple object instances, (2) it works on real data, (3) it works for partially occluded objects, and (4) because sparse keypoints are a computationally efficient representation for planning and control.

Now we introduce the progression of perception algorithms over time and highlight the novelty of our approach. Previous work in the field of computer vision includes algorithms for object classification, detection, pixel-wise segmentation, and more. Most recently, algorithms for instance segmentation have emerged such as Mask R-CNN (regional convolutional neural network) from Facebook (2). Given a single RGB image, Mask R-CNN predicts a class label (i.e. person, car, tree, dog, etc.) for each pixel. Furthermore, each

instance–group of pixels corresponding to the same class–is distinguished from one another. Mask R-CNN for instance segmentation is effective, but it alone lacks the detail needed for robotic manipulation. In particular, robot manipulation requires detail at a finer level to know where to grasp, which pixels (of images taken from different camera viewpoints) correspond to the same world points, etc. SparKey addresses this "detail" limitation by identifying a sparse set of unique 2D keypoints regardless of camera viewpoint.

A recent paper called Dense Object Nets (3) tries to attack this detail limitation problem as well. In particular, Dense Object Nets can transform an RGB image into a "dense descriptor" image. We illustrate an example of this in Figure 1. This dense descriptor representation is great for robot manipulation because the robot can consistently manipulate the same points of multiple objects of the same class. For example, the robot could grasp every shoe by the shoelaces by locating the pixels in dense descriptor space that are the most green–as shown in Figure 1. A k-nearest neighbors algorithm in the embedding space is used in practice.

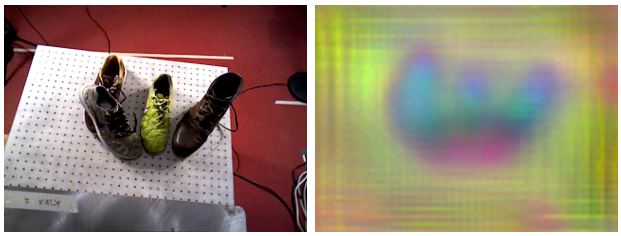

Figure 1: Here we show the output of Dense Object Nets (3). In this case, the network was trained to identify correspondences among shoes. Using network weights provided by Dense Objects Nets, we run inference for the RGB image to produce the output dense descriptor image (on the right). Visualized in color space, we can see that the same colors represent the same part of the shoes–even across different instances.

Dense Object Nets successfully addresses the detail limitation problem, but SparKey's "sparse approach" can be used for pose estimation and is more computationally efficient (because it provides a sparser representation for planning and control). In SparKey, we detect and track only the most important keypoints that are learned for optimal pose estimation. Because our work uses only a few keypoints (instead of all pixels corresponding to an object), we can more efficiently track objects during real-time object manipulation. Furthermore, it is much simpler to plan a trajectory for a few points on an object than to plan a trajectory for the entire pixel region in dense descriptor space. We illustrate this concept with a toy (shoe) example in Figure 2. It's computationally easier to track and plan a trajectory for four points than for all pixel points of the shoe (if it were represented as a dense descriptor image). In addition to this sparsity improvement, we also add instance segmentation to our pipeline–something that was not implemented in Dense Object Nets.

Figure 2: Here we show an example where the sparse set of keypoints are denoted by stars for a shoe. Clearly it is more efficient to track just these four points through 3D space than it is to track an entire point cloud of dense descriptor points from the Dense Object Nets (3) work. The same efficiency logic holds for planning and control (planning a trajectory through space and controlling it with robotic manipulation).

To learn these sparse keypoints, SparKey improves upon self-supervised techniques from both Dense Object Nets and KeypointNet (1). We use data from the work of Dense Object Nets. These data consists of RGB-D images of objects and known camera poses. We then use these real data with a modified network from KeypointNet. KeypointNet learns optimal keypoints for pose estimation, but it was used with synthetic data

and didn't work for occlusions. By combining aspects of Mask R-CNN for instance segmentation, data from Dense Object Nets, and techniques from KeypointNet, we present SparKey as a new perception pipeline for automatic keypoint detection and tracking through occlusions.

Note that we could have trained a network similar to OpenPose (4) (Figure 3) to detect keypoints in our objects. However, OpenPose requires human-labeled keypoints. We intentionally avoid human-in-the-loop techniques, as we believe that a self-supervised technique is both (1) more time efficient and (2) more precise and optimal for pose estimation. Humans make errors and do not have adequate judgment to decide which points are best for pose estimation; evidence of this is in the KeypointNet paper (1).

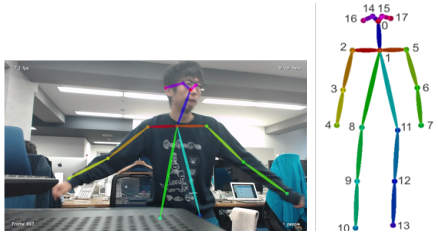

Figure 3: Here we show the output of OpenPose (4) when applied to humans (which it was built for). SparKey detects similar keypoints, but we apply these to arbitrary objects, not humans. Furthermore, unlike OpenPose, we use no human labels. We are effectively bringing the popular problem of (human) keypoint detection to an application in robotic manipulation.

2 Related Work

In this section, we look at existing methods for generating visual keypoints (features that can be located in RGB images taken from multiple camera viewpoints) and their relevance to robotic manipulation.

Research in the field of visual keypoints began with hand-crafted techniques. Algorithms such as Scale Invariant Feature Transform (SIFT) (5) were created for sparse keypoint tracking across consecutive images taken from a moving camera (or a moving object and a static camera). SIFT works by finding a sparse set of keypoints based on local gradients that are consistent across images. The sparse keypoints are invariant to viewpoint, which led to applications in image stitching, object recognition, 3D reconstruction, visual odometry, and more. Clearly SIFT has relevance to robotic manipulation (because it describes the state of the world in some capacity). However, hand-crafted algorithms are being replaced with "learned methods" based on various machine learning techniques (which utilize convolutional neural networks (CNNs)) (6).

Learned visual keypoint methods are proving to be more reliable than hand-crafted methods for robotic manipulation due to their robustness in different lighting conditions, ability to handle partial occlusions, and more efficient representations (3). Building off of work from Choy *et al.* (7) and Schmidt *et al.* (8), Dense Object Nets (3) has demonstrated that CNNs can effectively learn dense visual object descriptors for robotic manipulation applications. Dense Object Nets trained their contrastive loss Siamese network with self-supervised data collected automatically with a 7-DOF robot arm and an RGB-D sensor. Dense Object Nets is an improvement over other methods that rely heavily on human labeled data (7) and produce a meaningful "dense descriptor" result depicted in Figure 1. However, there are two limitations with Dense Object Nets (3) for robotic manipulation: no instance segmentation and no sparse representation.

Firstly, background subtraction is used instead of instance segmentation. In Dense Object Nets, it's currently impossible to identify multiple objects in the scene. Figure 1 illustrates this. Because the shoes overlap, background subtraction can no longer tell which pixels belong to which shoe. SparKey, however, identifies object instances. As far as we know, no work combines both instance segmentation and learned keypoints for robotic manipulation.

Secondly, the representation of Dense Object Nets for planning and control is computationally inefficient. It requires the planning and control system to track all pixels (because all pixels are keypoints) that correspond to the object of interest (a shoe, for example). We solve this issue by only tracking a sparse set of keypoints. This solutions makes computation faster for pose estimation and planning trajectories for keypoints through 3D space. A toy example of this concept is depicted in Figure 2.

Overall, our work transforms RGB images to sparse keypoints that are learned optimally for pose estimation. Keypoint detection was popularized by detecting joints of humans. For example, OpenPose by Simon *et al.* (4) has shown success for detecting human keypoints from RGB images in real-time (Figure 3). OpenPose, however, is trained with human-labeled data. Keypoint detection has recently been applied to other applications.

KeypointNet by Suwajanakorn *et al.* (1) is work that learns a sparse set of 2D keypoints that are optimized for pose estimation in RGB-D images. This requires no ground truth keypoint annotations–only camera poses and multiple images taken of a static scene. However, some limitations of this paper are its dependence on synthetic data from ShapeNet (9), no instance segmentation, and it not working with occlusions.

Our work aims to address the limitations of Dense Object Nets and KeypointNet. We eliminate the tedious annotation process by leveraging the Dense Object Nets (3) dataset and techniques from both Mask R-CNN and KeypointNet.

3 SparKey Pipeline Methods

SparKey is used for automatic keypoint detection and tracking objects through occlusions from RGB images ($\mathbb{R}^{W \times H \times 3}$). We use real-world images instead of synthetically rendered images for better accuracy in real robotic manipulation tasks. Utilizing techniques from Dense Object Nets (3) and KeypointNet (1), we create a pipeline to detect and track keypoints of objects for robotic manipulation. We improve on these works to focus on partially occluded objects.

Our pipeline is comprised of sequential tasks including automatic data collection, training Mask R-CNN for instance segmentation, automatic keypoint discovery for pose estimation, and we finally use a modified network from KeypointNet for keypoint detections through occlusions. We explain all methods in the context of robotic manipulation using the shoe object class. See a diagram of the components in Figure 4.

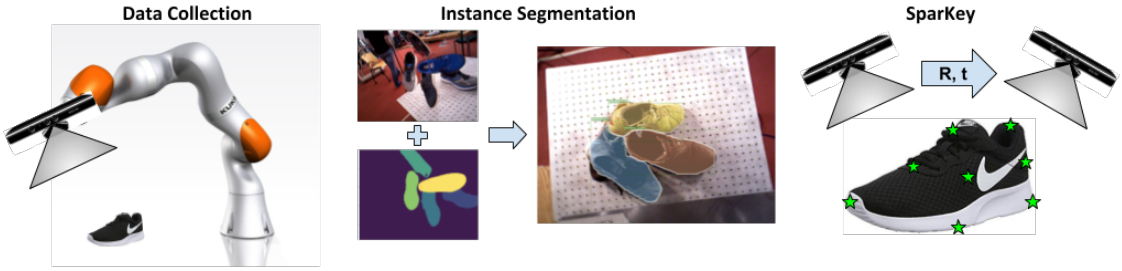

Figure 4: This is the overall methods pipeline for SparKey. The three major steps are shown left to right: data collection, instance segmentation, and the SparKey network.

3.1 Automated Data Collection

Previous work from Dense Object Nets (3) provides self-supervised methods to collect ground truth data from a RGB-D camera mounted at the end of a 7-DOF robot arm. An object is placed on the table in front of the robot, and the robot arm moves around taking RGB-D pictures from many angles. Using the forward kinematics of the robot arm, we know the pose T_{W_L} of the left camera L in the world frame W. Building off of and improving upon KeypointNet (1), our training data takes the form described in Section 3.3.

These data is all we need for automatic keypoint detection with SparKey. We explain how the data is used in Section 3.3. Techniques come from the KeypointNet paper, and we then make edits to optimize for occlusions and real data.

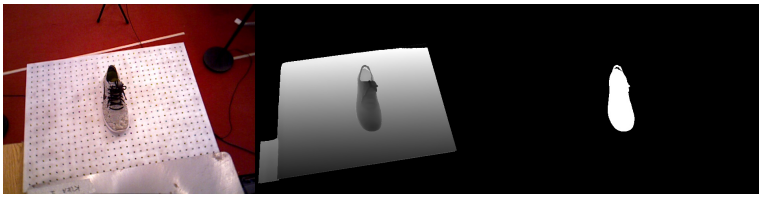

Figure 5: Here we show the data provided from the Dense Object Nets (3) work. We have RGB images (left) with corresponding depth images (middle) and background subtracted binary masks (right). We also have camera poses with respect to the world origin for each image.

The binary object masks, B_L and B_R, are computed by reconstructing the scene with static fusion from a sequence of RGB-D images taken from different viewpoints with known robot arm forward kinematics. We use the data from Dense Object Nets, which is shown in Figure 5. Using the constructed point cloud, everything but the object on top of the table can be perfectly removed and masks can be generated. Details on this can be found in the Dense Object Nets paper (3).

3.2 Synthetic Instance Segmentation

Instance segmented objects are used for training SparKey, so we must detect instances at runtime before passing the segmented images into SparKey for keypoint detection. We do this by training Mask R-CNN on our data to learn a mapping from RGB images to instance masks.

For the instance segmentation network, we use Facebook's Detectron (10) implementation of Mask R-CNN, which uses Caffe2 (11). Mask R-CNN takes as input an RGB image and regresses the following three items: bounding boxes, instance masks, and classifications. This process 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 (12), 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 (2). For the purpose of our work, we are most interested in the Mask R-CNN branch that predicts the mask for all object instances. We show a diagram from the original Mask R-CNN paper depicting its input and output in Figure 6.

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), as described in section 3.1. This works for single-shoe images, but when there are multiple shoes, such as in the RGB image in Figure 1, 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,

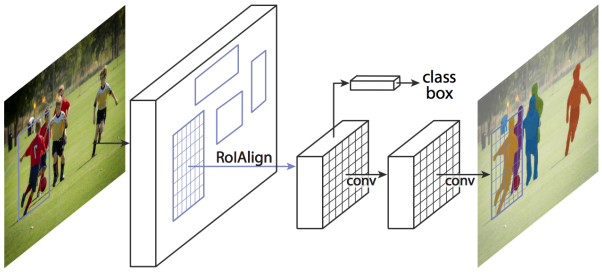

Figure 6: This is the framework for Mask R-CNN, which comes from the original paper (2).

we create an augmented dataset and train Mask R-CNN for instance segmentation.

3.2.1 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. Note that 7 is an arbitrary number that we found to produce accurate segmentation results. We then choose one of the images to be the background. Finally, we 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 pseudocode 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 to keep track of which pixels belong to which shoe;
end

Algorithm 1: Synthetic Image Merging

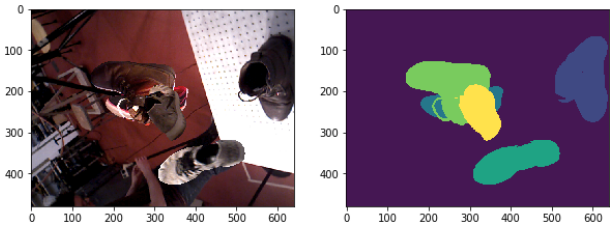

Figure 7: On the left we show a synthetic RGB image and on the right we show the corresponding masks. We've written code to store all the annotations in COCO format. This allows for good integration with the Mask R-CNN implementation we use.

3.2.2 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. During training, we initialize our network with pretrained weights on ImageNet.

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 results sufficient for using with the SparKey network. We show an example of instance segmentation results in Figure 8.

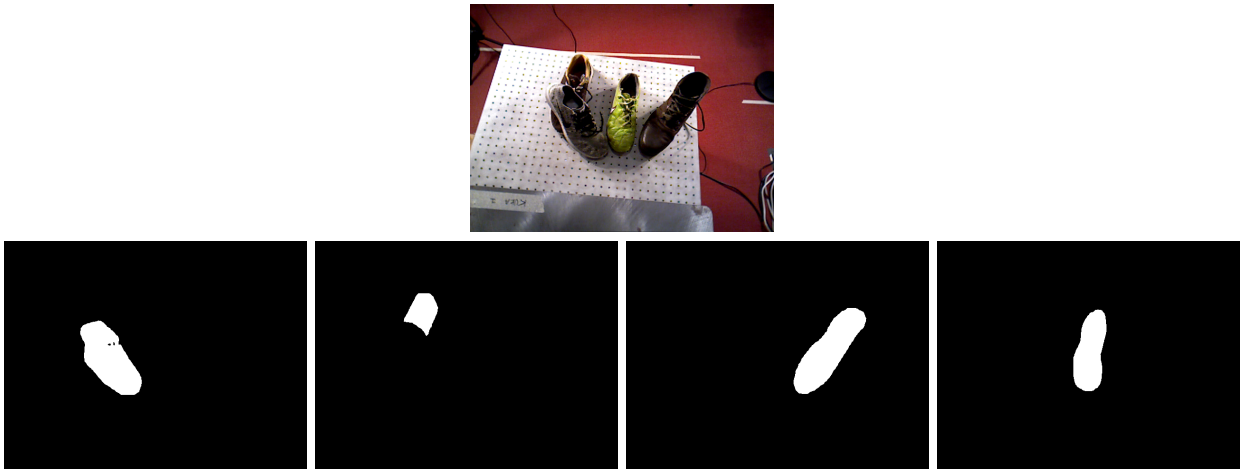

Figure 8: Mask R-CNN inference on the RGB images creates the binary masks shown here.

3.3 Automatic Keypoint Discovery

For automatic keypoint discovery, we start with the same techniques as KeypointNet (1). We get our data to work with the open-sourced TensorFlow implementation of KeypoinNet (13). Recall from before that we have the following data available to us:

- Left RGB Image: $I_L \in \mathbb{R}^{W \times H \times 3}$
- Left Depth Image: $D_L \in \mathbb{R}^{W \times H}$
- Left Binary Object Mask: $B_L \in \mathbb{R}^{W \times H}$
- Left Camera Pose: $T_{W\_L} \in \mathbb{R}^{4 \times 4}$
- Right RGB Image $I_R \in \mathbb{R}^{W \times H \times 3}$
- Right Depth Image: $D_R \in \mathbb{R}^{W \times H}$
- Right Binary Object Mask: $B_R \in \mathbb{R}^{W \times H}$
- Right Camera Pose: $T_{W\_R} \in \mathbb{R}^{4 \times 4}$
- Intrinsic Camera Matrix $K \in \mathbb{R}^{3 \times 3}$

At a high level, we use a segmented RGB image to predict n keypoints that have (u, v) pixel coordinates and depth z. Using the camera poses, we are able to project the predicted keypoints from one camera frame into the other. We can then look at the difference between the predicted keypoints and the keypoints in the other image. We optimize for pose estimation by putting a loss on relative rotation by using the Procrustes' alignment algorithm based on the keypoints predicted in each camera frame (1).

The network is fairly straightforward for regressing keypoints and depths, so we leave these details for the

KeypointNet paper (1). However, we go over the process of projecting one set of coordinates into the other frame here. They key idea is being able to predict keypoints for good pose estimation (rotation estimation) and making the keypoints the same world coordinates in each view.

Here we go over the steps of projecting the pixels from the left camera pose T_{W_L} to the right camera pose T_{W_R}. We first create the transform from the left camera's world frame to the right camera's world frame, which we define as T_{R_L}.

$$T_{R\_L} = T_{W\_R}^{-1} T_{W\_L}$$

We also have K, which is the camera intrinsic matrix. K is used for unprojecting points to world coordinates and projecting points into camera coordinates.

$$K = \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

The following are the steps for going from points in the left camera L to the right camera R. We use each of these intermediate steps to compute different losses for our network. Details are mentioned in the KeypointNet paper (1).

1. Left Camera Plane Coordinates, C_L
   $$C_L = \begin{bmatrix} u_L \\ v_L \\ 1 \end{bmatrix} z_L$$

2. Left Camera World Coordinates, W_L
   $$W_L = K^{-1} C_L = \begin{bmatrix} x_L \\ y_L \\ z_L \end{bmatrix}$$

3. Left Homogeneous Camera World Coordinates, H_L
   $$H_L = \begin{bmatrix} x_L \\ y_L \\ z_L \\ 1 \end{bmatrix}$$

4. Right Homogeneous Camera World Coordinates, H_R
   $$H_R = T_{R\_L} H_L = \begin{bmatrix} x_R w_R \\ y_R w_R \\ z_R w_R \\ w_R \end{bmatrix} = \begin{bmatrix} x_R \\ y_R \\ z_R \\ 1 \end{bmatrix} w_R$$

5. Right Camera World Coordinates, W_R
   $$W_R = \begin{bmatrix} x_R \\ y_R \\ z_R \end{bmatrix}$$

6. Right Camera Plane Coordinates, C_R
   $$C_R = K W_R = \begin{bmatrix} u_R z_R \\ v_R z_R \\ z_R \end{bmatrix} = \begin{bmatrix} u_R \\ v_R \\ 1 \end{bmatrix} z_R$$

We can project keypoints from the right camera to the left camera in a similar way. We now use each of these keypoints to optimize for pose estimation. Specific details are left to the KeypointNet paper (1). We explain our loss modifications to achieve good results on real data (Section 3.3.1) and how to make it work for occlusions (Section 3.3.2).

3.3.1 Real World Losses

One modification we add to KeypointNet is a new distance loss. The loss helps put the depth estimates in the right order of magnitude. One significant issue with using the KeypointNet methods is its sensitivity to real-world coordinates. More specifically, it's not agnostic to the scale of world coordinates used. We account for these issues by using a distance loss to help keypoints to be regressed in the correct range. We found that without the depth loss SparKey is quick to diverge in the first steps of training. See Figure 9 for an illustration of the depth loss. The following information is a replication of that shown in the Figure 9 image:

$L_{distance} = \sum_i^N \max(0, ||d_i - c_s||^2 - r_s^2)$
d_i = distance of point i
N = number of points
c_s = distance centroid for sequence s
r_s = sphere radius for sequence s

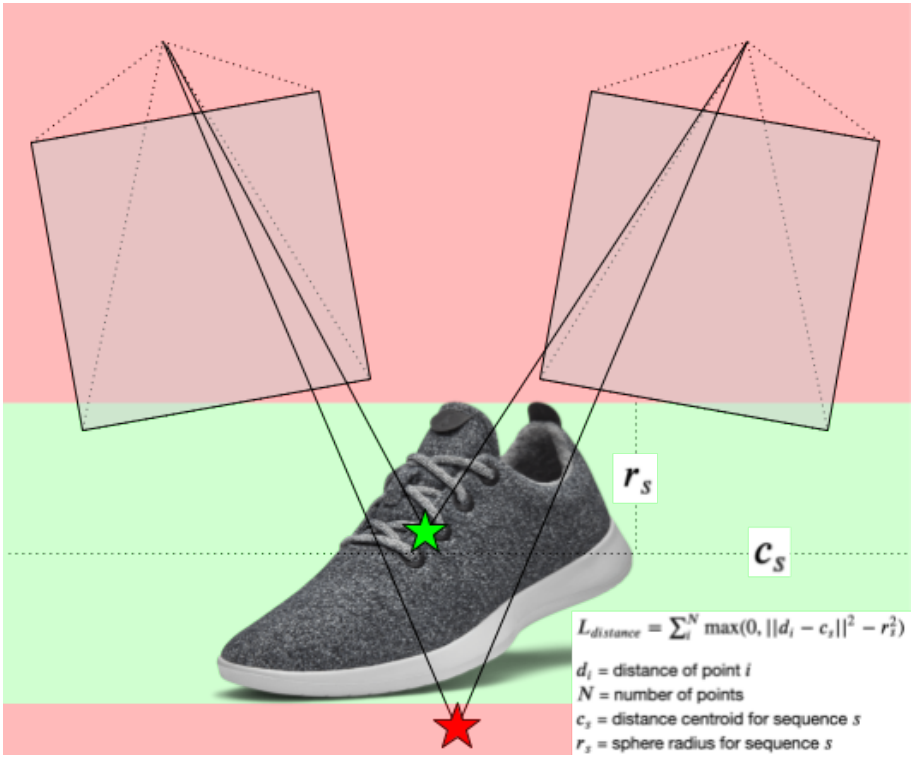

Figure 9: SparKey's depth loss.

Another modification we are also working on is a new separation loss. Currently we are using the same loss as KeypointNet (with a sufficiently chosen hyperparameter), but we are working on a better loss to normalize the world coordinates. This will create better results for real data because results won't depend on world coordinate scale.

3.3.2 Occlusion Aware Augmentations

KeypointNet only works for self-occlusions, meaning the keypoints are being occluded by the object itself. However, in robotic manipulation, objects will often be partially occluded by other objects or the robot gripper, for example. We solve this issue by synthetically blocking out part of the mask in the RGB images used in training. All losses remain the same, and by doing this modification during training, we are making SparKey more robust to occlusions. See Figure 10 for an illustration of the occlusion loss, and see Figure 11 for an actual image example.

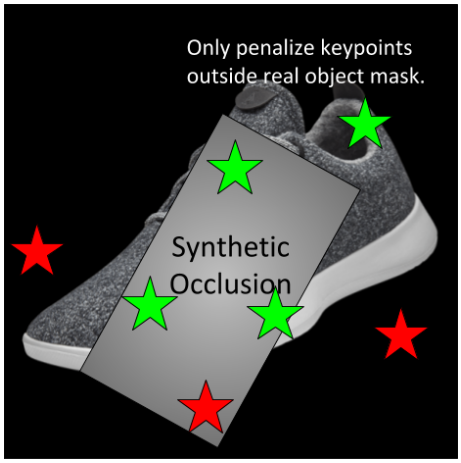

Figure 10: SparKey's occlusion loss.

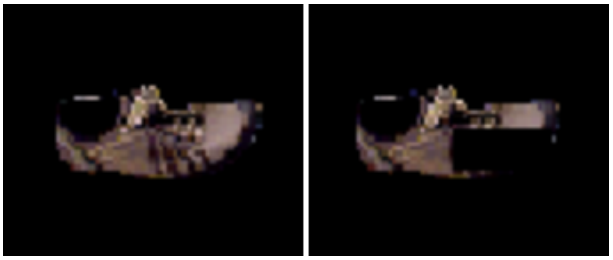

Figure 11: On the left is the original image crop, and on the right is the crop with a portion of the shoe occluded. However, we don't penalize for keypoints predicted in this occluded space.

3.4 Putting It All Together

We train our networks with the data collected and preprocessed for the specific algorithm at hand. During robotic manipulation, we use instance segmentation to isolate the object instances. We then feed the masked RGB image through the SparKey network. The final result is keypoints with both location and depth, which allows us to project the points into the camera frame and do pose estimation. We can do this for all masks and therefore all instances. Our methods also work for partially occluded objects due to our new loss described in Section 3.3.2.

3.4.1 Generalizability of Techniques

All of the techniques used here are for robotic manipulation, but we use all the same techniques for augmented reality (AR) applications. In particular, if a phone scanned an object on top of a surface with a known background, some SLAM (Simultaneous Localization and Mapping) algorithm should be able to reconstruct the scene and create masks by subtracting the background. We can then create a dataset as described earlier. This automatic discovery would be extremely useful in tracking objects for AR applications. The instance segmentation with Mask R-CNN could be done in a similar way given a sufficient amount of data.

4 Evaluation and Results

For the evaluation of SparKey, we focus on the individual components of the pipeline and look at keypoint detection and pose estimation results for shoes. After the instance segmentation component, SparKey was trained with a dataset of shoes taken from 29 unique video sequences from Dense Object Nets. Each TFRecord contains data points described in the methods (Section 3). Table 1 explains the dataset that we used for the results. Note that "pairs" refers to a left and right image pair, which makes up a data point. Also, we predict for 10 unique keypoints in all keypoint detection scenarios.

SparKey Dataset			
	Training	Test	Validation
Num TFRecords	6	2	2
Pairs per TFRecord	500	500	500
Total pairs	3000	1000	1000

Table 1: This is the breakdown of the data we used for SparKey. Each "pair" is a left and right image with corresponding data described in the methods section.

4.1 Instance Segmentation Results

Here we describe the results of the Mask R-CNN (instance segmentation) component of SparKey. After creating synthetic data with COCO formatted labels and training Mask R-CNN, we obtained results shown in Figure 12. These results used a training set of 10,000 images and 2,000 validation images (with ground truth annotations like those shown in Figure 7). Our metrics on the 2,000 validation images are shown in Table 2 (bounding box) and Table 3 (segmentation). These values are summarized in the COCO documentation (14).

Recall that the output of instance segmentation will be used as input to the keypoint detection components of SparKey, described in the following sections. The numbers shown in the tables are quite good, but note that the validation set contains only synthetic images. It's impossible to generate ground truth labels for real images of overlapping objects, but we still obtain good empirical results, as depicted in Figure 12.

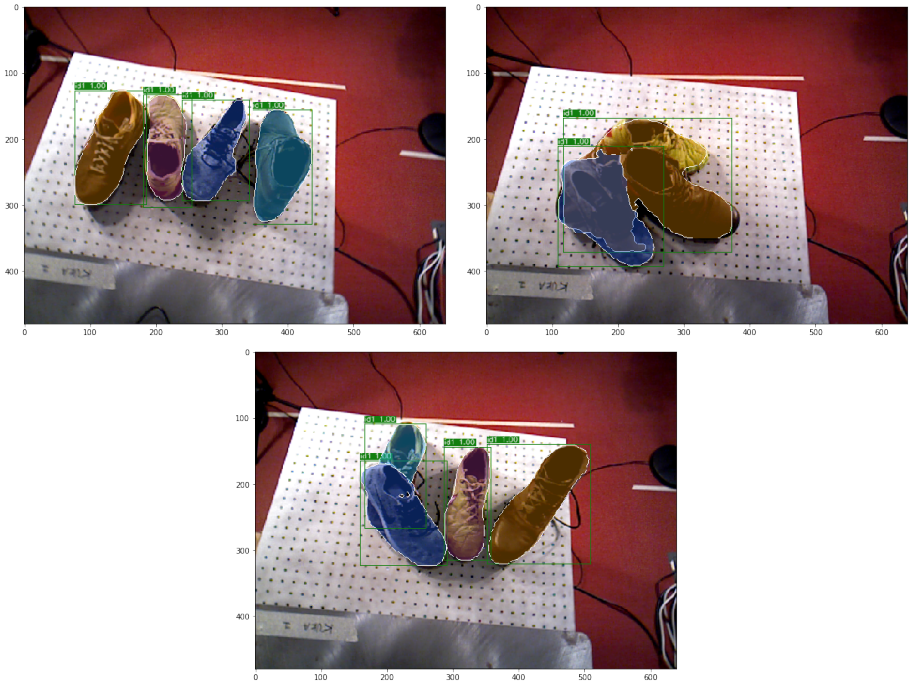

Figure 12: Here are some examples of using Mask R-CNN for inference on RGB images with multiple shoes. The predicted instance masks are overlaid on the images.

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 2: Bounding Box 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 3: Segmentation Evaluation Metrics on Validation Set

4.2 Depth Loss

KeypointNet does not include a depth loss, but we realized that it's extremely important for SparKey during training and gradient descent. Table 4 shows this where we have an identical network trained on our SparKey dataset with either **depth loss** or **no depth loss**. A typical training session run for 200K iterations, but 5K is plenty to show the importance of the depth loss.

With depth loss, all depth predictions fall in the correct range by iteration 3K of training. However, without penalizing the predicted depth, the depth loss rises very quickly. This observation suggests that the network needs loss on depth in order to make depth predictions on the right order of magnitude. Pose estimation and multi-view consistency both require good depth predictions, so this loss only helps SparKey achieve good results as fast as possible. It's unclear to us why no depth loss existed in KeypointNet.

Depth Loss after Number of Iterations (Training Set)			
	250	3K	5K
With Depth Loss (19-04-08_08-09-43)	206.8	0	0
Without Depth Loss (19-04-15_10-03-19)	4.99E+05	6.66E+07	1.89E+08

Table 4: This table shows the depth loss after iterations of training with and without it being penalized. SparKey appears to not learn depth from the other losses, such as those for pose estimation and multi-view consistency, as quickly as with an explicit depth loss.

4.3 Separation Loss

As said in the KeypointNet paper, the separation loss is very important to avoid singularities where multiple keypoints collapse into one. Figure 13 illustrates this. The left image has a sufficient separation loss, where the keypoints are separated. The right image, however, is trained with an insufficient separation loss, where only three of the 10 keypoints are unique. This result makes sense because only three points are technically required for pose estimation.

A significant problem with the KeypointNet paper is that the separation loss is chosen very specific to the dataset. Currently we are left with the same problem, where we must choose hyperparameters very carefully and based on the magnitude of our dataset world coordinates. We are planning to solve this problem by choosing the hyperparameters in a normalized way to penalize closeness of keypoints and reward spaced out

keypoints. This is currently in the works, but for now we use a sufficient hyperparameter.

See Table 5 for results with and without separation loss. Note that the **without** case is equivalent to an insufficient parameter situation.

Penalizing Separation Loss after 200K Iterations			
Experiment	Multi-View (Validation)	Pose (Validation)	Separation (Training)
With Sep Loss (19-04-10_04-29-05)	0.05298	0.599	1.00E-03
Without Sep Loss(19-04-10_04-30-07)	0.04969	0.5996	5.27E-03

Table 5: This table shows that the separation loss remains high if not penalized, even if multi-view consistency and pose estimation work well. This is shown in the images in Figure 13 where multiple keypoints map to the same location.

Figure 13: The left image uses a network with a good separation loss; the right image does not. These correspond to the experiments in Table 5. More points are unique in the left compared to the right. Each network should be producing 10 unique keypoints if done correctly.

4.4 Pose Estimation

Pose estimation is arguably the metric we care most about, as it implicitly encodes multi-view consistency as well. Furthermore, pose estimation is the result that can be directly fed into a planning and control pipeline for robotic manipulation tasks. In Table 6, we show some results of varying the pose loss, noise, and learning rate (LR).

The pose loss is simply the magnitude of the penalty (relative to the other losses). The noise is the random noise we add to the points before estimating the rotation used when computing the loss. The learning rate was also varied to see if speed of training would change. Based on Table 6, our best results were found for experiment 19-04-12_01-53-40. Because pose and multi-view metrics were best for this experiment, we designate experiment 19-04-12_01-53-40 as our best result. The full hyperparameters for this log are in Table 8.

SparKey Sensitivity Ablation Study							
Experiment	Loss Weightings					Validation Results (200K)	
	Pose	Multi-View	Separation	Noise	LR	Pose	Multi-View
19-04-12_01-53-40	1.0	10.0	100.0	0.01	0.001	0.2084	5.24E-03
19-04-12_01-55-01	1.0	10.0	100.0	0.001	0.01	0.3304	0.03765
19-04-12_01-57-59	2.0	10.0	100.0	0.01	0.01	0.4066	0.01668

Table 6: A sensitivity study with different weightings. This table shows how important it is to choose the hyperparameters. Note that the noise also changes results a lot. Also, the LR seems to speed up training in this case.

4.5 Occlusions

We used occlusion augmentations with the best hyperparameters found from experiment 19-04-12-01-53-40. In our augmentation experiments, there was a 25% chance that we would occlude $\frac{1}{4}$ of given shoe. Using the augmentations, our results did not improve and were fairly equal. However, we believe this may be due to lack of occluded images seen in evaluation images. See http://sparkey.xyz/ for a direct qualitative experiment of these experiments (with and without occlusion augmentations). Experimenting with more occlusion augmentations during training will likely result in better results.

4.6 Multi-Class Consistency

Because our training set consists of multiple shoe types, we are able to achieve multi-class consistency and good generalization. See Figure 14 for visual results. These images were not necessarily seen during training; they are instead pulled directly from the images of Dense Object Nets data. The colored keypoints should be in roughly the same location across images. More results can be seen at our project website, http://sparkey.xyz/. The website includes video data, which convey consistency results much better.

Dissecting Figure 14, we use Table 7 to empirically rank which Dense Object Nets (DON) logs that SparKey performed best on. Results are largely a function of both what is seen during training and how different the shoe is from the others. Based on our own inspection, we rank the results 1 (best) to 5 (worst). See Table 7 for more details.

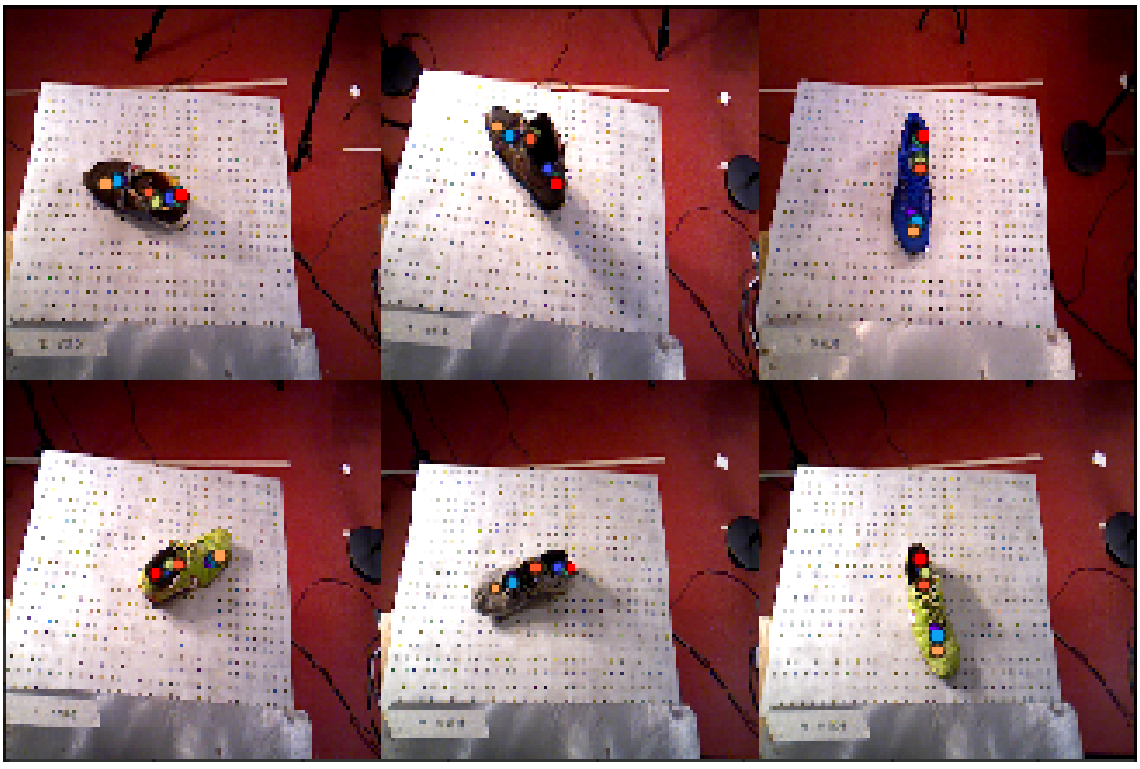

Figure 14: Here we show images from the logs described in Table 7. Logs are sorted top left to right, then bottom left to right.

Visual Results with Experiment 19-04-12_01-53-40		
DON Log	Type of Shoe	Empirical Ranking
2018-05-14-22-10-53	Red Nike	4
2018-05-14-22-57-40	Boot	5 (worst)
2018-05-09-18-19-27	Blue Shoes	1 (best)
2018-05-15-00-04-06	Yellow Cleats (1)	3
2018-05-15-00-26-19	Gray Nike	2
2018-05-15-00-06-46	Yellow Cleats (2)	3

Table 7: Here are some empirical results for the different types of shoes used during evaluation. 1 is best and 5 is worst (because of the repeated Yellow Cleats class).

Best Hyperparameters (Experiment 19-04-12_01-53-40)
num_filters=64, # Number of filters.
num_kp=10, # Number of keypoints.
loss_pose=1.0, # Pose Loss.
loss_con=10.0, # Multi-view consistency Loss.
loss_sep=100.0, # Separation Loss.
loss_sill=2.0, # Silhouette Loss.
loss_variance=0.5, # Variance Loss (part of Silhouette loss).
loss_depth=1.0, # Depth Loss.
sep_delta=0.001, # Separation Loss (depends on world coordinates).
noise=0.01, # Noise added during estimating rotation (depends on world coordinates).
learning_rate=1.0e-2,

Table 8: Experiment 19-04-12_01-53-40 had the best hyperparameters, which are depicted here.

4.7 Real-Time Inference Performance

Using an NVIDIA GeForce RTX 2080 Ti GPU, we can achieve a runtime of a single image in 3.044 milliseconds, which was averaged over 1000 images. This is using a batch size of 1 for each of the 1000 images. Mask R-CNN is comparatively fast. The combined pipeline of SparKey is fast enough to run real-time for pose estimation in our perception pipeline for any typical robotic manipulation task.

5 Conclusion

In conclusion, our perception method improves upon the Dense Object Nets (3) paper by only locating a sparse set of 2D keypoints which is more computationally efficient for tracking (pose estimation) and planning and control for robotic manipulation. To create SparKey, we use the Dense Object Nets dataset and make improvements to the KeypointNet (1) methods. After adding instance segmentation to the Dense Object Nets pipeline, we are able to find the optimal 3D keypoints for pose estimation. SparKey is more computationally efficient and provides a better representation for robot manipulation than existing methods. Furthermore, because we solve the instance segmentation problem, we can use the detected sparse keypoints in planning and control algorithms for multiple objects.

References

[1] S. Suwajanakorn, N. Snavely, J. Tompson, and M. Norouzi, "Discovery of latent 3d keypoints via end-to-end geometric reasoning," 2018.

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

[3] 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.

[4] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017.

[5] C. Liu, J. Yuen, and A. Torralba, "Sift flow: Dense correspondence across scenes and its applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 978–994, May 2011.

[6] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, p. 640–651, Apr 2017.

[7] C. B. Choy, J. Gwak, S. Savarese, and M. Chandraker, "Universal correspondence network," 2016.

[8] T. Schmidt, R. Newcombe, and D. Fox, "Self-supervised visual descriptor learning for dense correspondence," *IEEE Robotics and Automation Letters*, vol. 2, pp. 420–427, April 2017.

[9] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "Shapenet: An information-rich 3d model repository," 2015.

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

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

[12] 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.

[13] "Tensorflow keypoinnet." https://github.com/tensorflow/models/tree/master/research/keypointnet.

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

[15] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. J. Guibas, "Normalized object coordinate space for category-level 6d object pose and size estimation," 2019.

[16] I. K. Riza Alp Güler, Natalia Neverova, "Densepose: Dense human pose estimation in the wild," *arXiv*, 2018.

[17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *Lecture Notes in Computer Science*, p. 740–755, 2014.