

A DEEP 3D OBJECT POSE ESTIMATION FRAMEWORK
FOR ROBOTS WITH RGB-D SENSORS

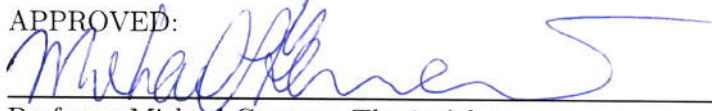
by

AMEYA WAGH

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Robotics Engineering
by


MAY 2019

APPROVED:



Professor Michael Gennert, Thesis Advisor



Professor Emmanuel Agu



Professor Berk Calli

Abstract

The task of object detection and pose estimation has widely been done using template matching techniques. However, these algorithms are sensitive to outliers and occlusions, and have high latency due to their iterative nature. Recent research in computer vision and deep learning has shown great improvements in the robustness of these algorithms. However, one of the major drawbacks of these algorithms is that they are specific to the objects. Moreover, the estimation of pose depends significantly on their RGB image features. As these algorithms are trained on meticulously labeled large datasets for object’s ground truth pose, it is difficult to re-train these for real-world applications.

To overcome this problem, we propose a two-stage pipeline of convolutional neural networks which uses RGB images to localize objects in 2D space and depth images to estimate a 6DoF pose. Thus the pose estimation network learns only the geometric features of the object and is not biased by its color features. We evaluate the performance of this framework on LINEMOD dataset, which is widely used to benchmark object pose estimation frameworks. We found the results to be comparable with the state of the art algorithms using RGB-D images. Secondly, to show the transferability of the proposed pipeline, we implement this on ATLAS robot for a pick and place experiment. As the distribution of images in LINEMOD dataset and the images captured by the MultiSense sensor on ATLAS are different, we generate a synthetic dataset out of very few real-world images captured from the MultiSense sensor. We use this dataset to train just the object detection networks used in the ATLAS Robot experiment.

Acknowledgements

I would like to thank Professor Michael A. Gennert, my thesis advisor for his valuable contribution throughout the journey of this thesis. I would like to thank Professor Emmanuel O. Agu and Professor Berk Calli for being part of the thesis committee. I would also like to thank Vinayak Jagtap for his guidance and support throughout the research. Lastly, I would like to thank my parents and my friends for the motivation and valuable support required throughout.

Contents

List of Figures	vi
List of Tables	viii
1 Introduction	1
1.1 Problem Statement	1
1.2 Related Work	2
1.3 Contribution	3
1.4 Outline	3
2 Preliminary Experiments	4
2.1 Background	4
2.1.1 Understanding Point cloud	4
2.1.2 Understanding RGB-D Image	5
2.2 Preliminary Experimentation	6
2.2.1 Approach 1: Point cloud only	6
2.2.2 Approach 2: RGBD	9
2.2.3 Conclusion	10
3 Methodology	11
3.1 System Overview	11
3.2 2D Image Segmentation	13
3.2.1 Understanding Image segmentation	13
3.2.2 SegNet	14
3.2.3 Implementation of 2D segmentation network	15
3.3 Object Pose estimation	16
3.3.1 Understanding Object Pose Estimation	16
3.3.2 Implementation of Pose Estimation Network	20
3.4 Loss Function	23
3.4.1 2D segmentation loss	23
3.4.2 Pose Estimation Loss	26

4	Experimentation and Results	27
4.1	Experimentation and Evaluation on LINEMOD dataset	27
4.1.1	LINEMOD Dataset	27
4.1.2	Data augmentation	28
4.1.3	Training	29
4.1.4	Segmentation Results	29
4.1.5	Pose Estimation Results	31
4.2	Experimentation and Evaluation on ATLAS Robot	36
4.2.1	Acquiring Real World Images	36
4.2.2	Acquiring Depth Images	40
4.2.3	TOUGH	42
4.2.4	ROS nodes	42
4.2.5	Drill Detector Task	46
4.2.6	Drill Pick-up Task	48
5	Conclusion	51
5.1	Conclusion	51
6	Future Work	53
A	Appendix	54
A.1	TOUGH API description	54
	Bibliography	58

List of Figures

2.1	Point cloud of a coffee mug	4
2.2	Sample image and its corresponding depth image	5
2.3	Object detection using feature histograms	7
2.4	Viewpoint Feature Histogram of bowl and mug	8
2.5	Visualization of predicted labels for objects in scene	8
2.6	Segmentation masks projected on RGB images	9
2.7	Object proposals in 3D space	10
3.1	Proposed 2-stage pipeline consisting segmentation stage and pose estimation stage. Q in the figure stands for quaternion vector of dimension 4x1 and T stands for translation vector of dimension 3x1.	12
3.2	A sample image showing semantic segmentation and instance segmentation [38]	13
3.3	Architecture of Segnet [1]	14
3.4	An RGB Image and a binary mask inferred from Segmentation network . . .	15
3.5	Pose estimation using EPnP [29]	18
3.6	Registration of point clouds using Iterative Closest Point [13]	19
3.7	Network architecture for object pose estimation	20
3.8	Depth image (top left), its JET encoding representation (top right) and image after element-wise multiplication (bottom)	21
3.9	Representation of object center in u,v coordinate space	22
4.1	Image of Ape from Linemod dataset (left), corresponding segmentation masks(right)	28
4.2	Sample image and its augmentations used during training	29
4.3	RGB images of ape, drill and bench-wise (left), their corresponding masks (Right).	31
4.4	Predicted and ground truth pose overlaid on the RGB images. The axes with a blob at the tip is the ground truth pose, the other is the predicted pose	33
4.5	Trends of accuracy with change in distance threshold for (from top left to bottom right) ape, benchwise, cam, can, cat, drill	34
4.6	Trends of accuracy with change in distance threshold for (from top left to bottom right) duck, egg-box, glue, hole-punch, iron, lamp	35
4.7	Images of the object captured with MultisenseSL	37
4.8	Ground truth masks for the real object images	37

4.9	Images of the Background captured with MultisenseSL	38
4.10	Synthetically Generated Images	38
4.11	Masks for Synthetically Generated Data	39
4.12	MultiSense SL sensor used on the ATLAS robot [33]	41
4.13	TOUGH API overview [9]	42
4.14	Drill detection on Atlas Robot	45
4.15	Drill detection on Atlas Robot	46
4.16	Sequence of images from a video of pose estimation experiment	47
4.17	From Top left to bottom right: object detected by the robot, planning and executing trajectory to grab the object, closing the gripper to grasp, returning to default pose	48
4.18	Failing to continuously detect object with 10DoF planning experiment	49

List of Tables

4.1	Segmentation Performance on LINEMOD Images	30
4.2	Statistics of Segmentation Performance on LINEMOD Images	30
4.3	Inference Speed on LINEMOD dataset	36
4.4	Comparative Analysis of dataset distributions	39
4.5	Segmentation Performance on Real world Images	40
4.6	Statistics of Segmentation Performance on Real world Images	40

Chapter 1

Introduction

1.1 Problem Statement

The problem statement considered in this thesis is to develop a perception framework for robots which do not have sensors such as Kinect. A lot of algorithms being trained are sensor specific or object specific. These algorithms use color keypoint or features to match specific templates. The network gets biased to these features, thus making it specific to the object. On the contrary, depth images encode only the geometrical information of the object and can be used to predict the pose hypothesis. Secondly, deep neural networks are resource intensive and thus, making it difficult to implement in real-time robotics applications.

In this work, different methods of object detection and pose estimation are explored and a convolutional neural network based architecture is proposed which uses RGB features to localize the object and then uses geometric features from the depth image to predict the object pose. First, the framework is trained and evaluated on a publicly available LINEMOD dataset. Then, the object localization module of the same network is retrained on fewer images for a particular object and tested on a real robot with a different sensor for object pick-up task. The pose estimation network is not retrained in this experiment.

1.2 Related Work

The problem of recognizing an object and estimating its pose has been a topic of interest for years. This has a lot of applications in robotics, mainly in the manipulation of the object. There has been an exponential growth in research on processing 3D point cloud scenes. Conventional computer vision techniques do not work out of the box on point clouds. Hence, new methods for feature description and segmentation are being proposed.

Traditionally, template matching techniques are used, which compute correspondences between a 3D model and a point cloud scene [13, 3, 16, 18]. These correspondences are then scored using a similarity function and thus a pose hypothesis with the best score is chosen. These algorithms are either sampling based or iterative in nature which is inefficient in inference speed. Similarly, template matching techniques are used to match the projections of 3D models on 2D images [32], [20], [37], [23], [24], [27]. Although template matching techniques show great performance in estimating object pose, they heavily depend on image key points or features. With occlusions, the numbers of key points reduce and thus affects these methods significantly.

Alternately, feature based methods perform better in handling occlusions. They are used to compute local and global features, which act as a signature of the object of interest. These features are then matched with a 3D model to find the correspondences in the image. Methods such as DLT [28] and E-PnP [29] are used traditionally to estimate a pose hypothesis by computing a relation between the 2D-3D key points.

With the massive success of Deep learning framework and increased precision in laser sensors, some end to end object recognition pipelines were proposed. Pose-CNN [40] is one such end to end network which uses Convolution neural networks and attempts to regress pose using just RGB images. Similarly, Deep-6D Pose [6] extends the Mask RCNN [14] framework to localize instances of objects and regress pose. IPose [19] uses similar instance segmentation network and leverages the depth image to obtain the object pose using geometrical methods. BB8 [32] uses a multilevel detection process in which they use VGG Network to detect object ROI and train a CNN on 2D projections of 3D objects to predict object pose. Another Multi-view approach by [42] uses multiple viewpoints to fit

pre-scanned 3D models to 2D segments of the object of interest.

1.3 Contribution

- Explored 2 approaches, a point cloud based approach and RGB-D based approach for detecting and localizing objects in 3D scenes.
- Proposed a 2 stage pipeline using Convolutional Neural Networks to infer 6 DoF pose of the object from RGB-D images
- Contributed to T.O.U.G.H, a C++ API library for humanoids research (main contributions are on tough perception package)
- Evaluated the proposed pipeline on LINEMOD dataset
- Experimentally validated the performance of the method for real-world application by implementing on ATLAS Robot.

1.4 Outline

The thesis is outlined as follows

- **Chapter 2** explores the point cloud based and RGB-D based approaches and builds a path for the proposed pipeline.
- **Chapter 3** gives an overview of each block of the proposed framework and dives deep into the implementation.
- **Chapter 4** evaluates the proposed framework on LINEMOD dataset and ATLAS robot.
- **Chapter 5,6** concludes the thesis and contemplates the future work.

Chapter 2

Preliminary Experiments

2.1 Background

This section gives an overview of 2 different representation of 3D information. The point cloud representation and RGB-D representation is explained briefly in the following subsections.

2.1.1 Understanding Point cloud

Point clouds are a representation of data points of an object in 3D-space. Every point on the object is represented by the x,y,z coordinates with reference to the sensor frame.

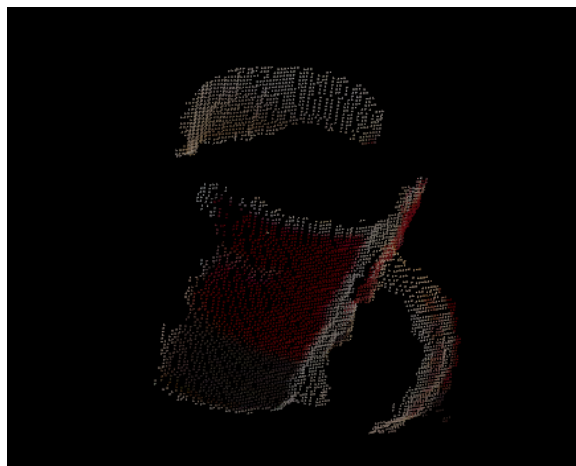


Figure 2.1: Point cloud of a coffee mug

As every individual point represents its position in the 3D space, a set of points describing a point cloud can be represented as Organized point cloud and Unorganized point cloud. An organized point cloud is represented as a 2D array similar to an image. This gives an advantage of computing graph based operations such as finding the nearest neighbor simpler and intuitive. Alternately, unorganized representation of point cloud structure is an array of points with their x,y,z values. Unorganized point cloud usually has dimensions such as $N \times 3$ where N is the number of points and 3 is the dimension for $[x,y,z]$ position vector. Moreover, points can also encode their color information and thus every point can be represented as $[x, y, z, R, G, B]$. Figure 2.1 shows a visualization of unorganized colored point cloud.

2.1.2 Understanding RGB-D Image

RGB-D is yet another representation of 3D objects. The RGB stands for the 3 color channels (RED, GREEN, BLUE) in the image. An additional depth channel is added to create a 4 channel image. Every pixel in the depth channel corresponds to the distance of the point from the image plane. In most cases, x and y -axes are on the image plane and z -axis is normal to the image. Thus, the z -values of the points are used to generate the depth image.



(a) RGB Image

(b) Depth Image

Figure 2.2: Sample image and its corresponding depth image

Figure 2.2 shows an instance of the RGB image and its corresponding image. The depth image is normalized to $[0-255]$ values for the ease of visualization in grayscale.

2.2 Preliminary Experimentation

To evaluate the feature-based method, the presented experiments consists of 2 approaches. The first approach uses only the point cloud representation of the object and the second uses RGB-D representation. These experiments assume the use case of table-top scenarios and dense point clouds.

2.2.1 Approach 1: Point cloud only

For the preliminary experimentation, multiple point cloud based approaches were tested on RGB-D Dataset [21]. To identify the objects in the scene, the point cloud scene was segmented into smaller point clouds using euclidean distance clustering. For every segment obtained, a global feature histogram was computed. Viewpoint Feature Histogram Estimator (VFHE)[35] is one of the techniques used to compute global features of a point cloud. The VFHE computes normals to every single point in the point cloud and angle of the normal with respect to the viewpoint vector. These angles are binned in 360 bins to form a feature vector. The RGB-D dataset [21] provides point cloud models of individual objects in the scene whose VFH can be used as ground truth. These histograms were used to train a Support Vector Machine (SVM)[15] classifier. At inference, this trained SVM was used to predict class labels for the segments obtained by euclidean distance clustering, see figure 2.3.

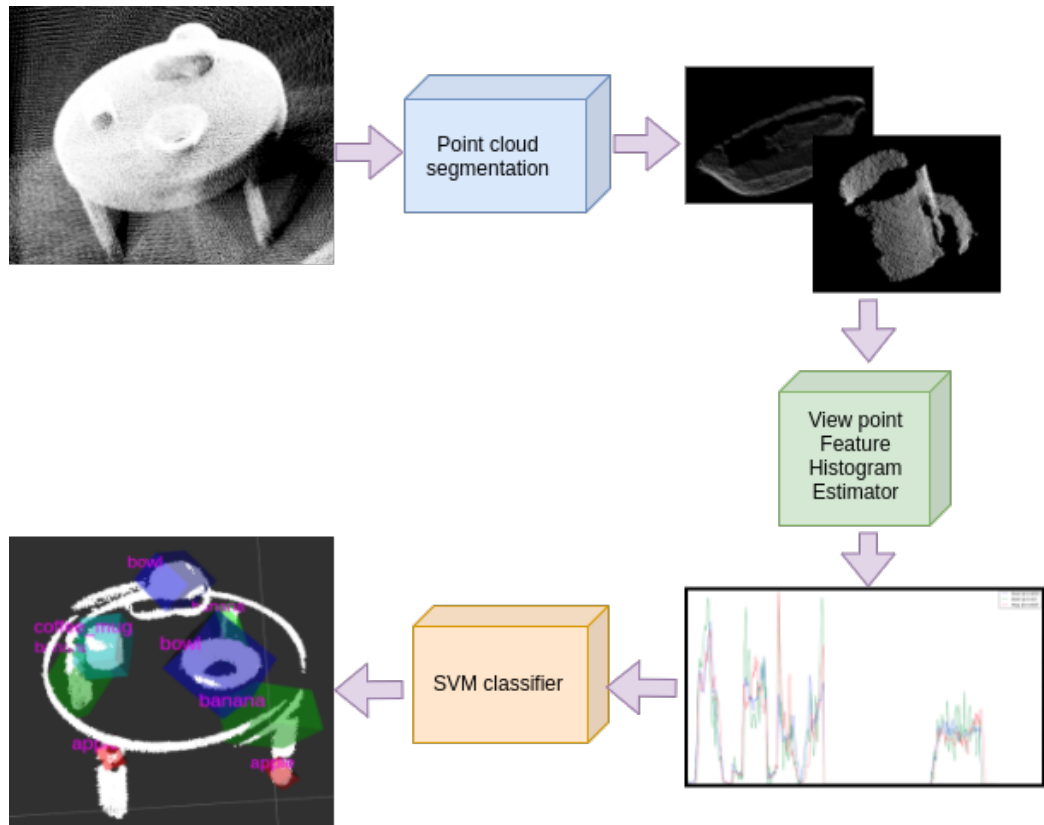


Figure 2.3: Object detection using feature histograms

It was observed that symmetric objects such as mugs and bowls have similar feature histograms which confused the SVM classifier, refer to figure 2.4 . Moreover, it took approximately 1-2 minutes to process the complete point-cloud on an i7 CPU with 16 GB Memory.



Figure 2.4: Viewpoint Feature Histogram of bowl and mug

The final results contained a lot of outliers due to over-segmentation and objects with symmetric geometry. Figure 2.5 shows the misclassified labels, where the edge of the table is detected as banana and the upper part of a cap is detected as a bowl.



Figure 2.5: Visualization of predicted labels for objects in scene

2.2.2 Approach 2: RGBD

Another approach with 2D segmentation proposals was tested 2.6 which performed better on symmetric objects. A SegNet [1] model pre-trained on ImageNet[5] was used to detect 2D segmentation masks, which were used as region proposals. These masks were then projected onto the aligned depth images to obtain a crop of the depth image. Using the camera intrinsics provided, this depth image was projected back into 3D space, reconstructing point clouds of the objects 2.7.



Figure 2.6: Segmentation masks projected on RGB images

Despite noise in the depth image, the point cloud generated closely resembled the actual object and the detection was significantly faster.

Figure 2.6 shows the RGB images and segmentation masks overlaid on the image.

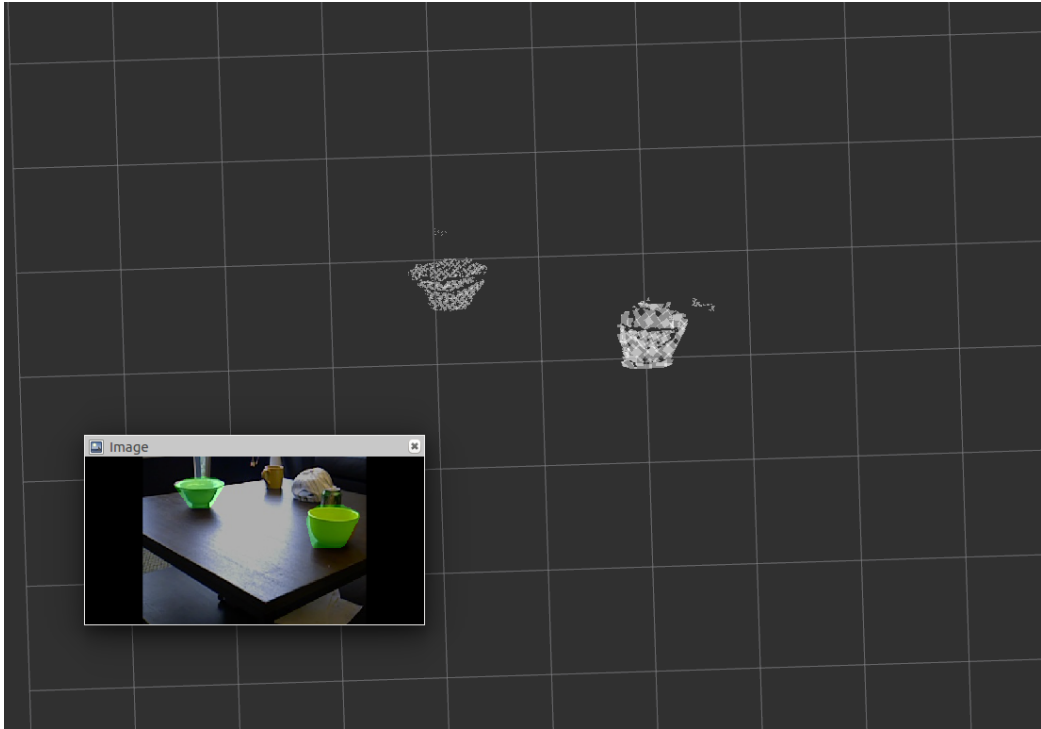


Figure 2.7: Object proposals in 3D space

The segmentation mask is a binary image which indicates pixels belonging to the object and background, whereas the depth image contains the distances of individual points from the image plane. By performing element-wise multiplication of the depth image with the segmentation mask, only the pixels corresponding to the objects have non-zero depth values. Using the camera intrinsic matrix, these depth images are converted into point clouds. Figure 2.7 shows a visualization of the reconstructed point cloud in Rviz software [10].

2.2.3 Conclusion

From the above experiments [2.2.1, 2.2.2], It can be concluded that the projection based method performs better than the point cloud only method. The RGB images play an important role in localizing the object. Based on these results, a 2-stage object pipeline is proposed which can localize objects and estimate their 6DoF pose. The RGB images are limited to the segmentation stage as the segmentation networks can be trained for new objects using techniques such as transfer learning.

Chapter 3

Methodology

3.1 System Overview

The object detection and pose estimation pipeline is divided into two stages, an RGB image segmentation network and an object pose estimation network. The object segmentation network is a more generic module and can be swapped with any state of the art binary image segmentation convolutional neural network. The contribution of this research is mainly towards developing a light-weight Convolutional Neural Network which predicts object pose as a pair of a position vector and a quaternion vector. As the framework is implemented on a machine connected to the actual robot with limited computational power (6GB VRAM), two large networks on Single GPU could not be accommodated. Thus, the model architecture is limited to smaller networks which agree with the inference speed and accuracy trade-offs. These computation specifications are only limited to the inference stage while the training is done on a large cluster of CPUs and GPUs.

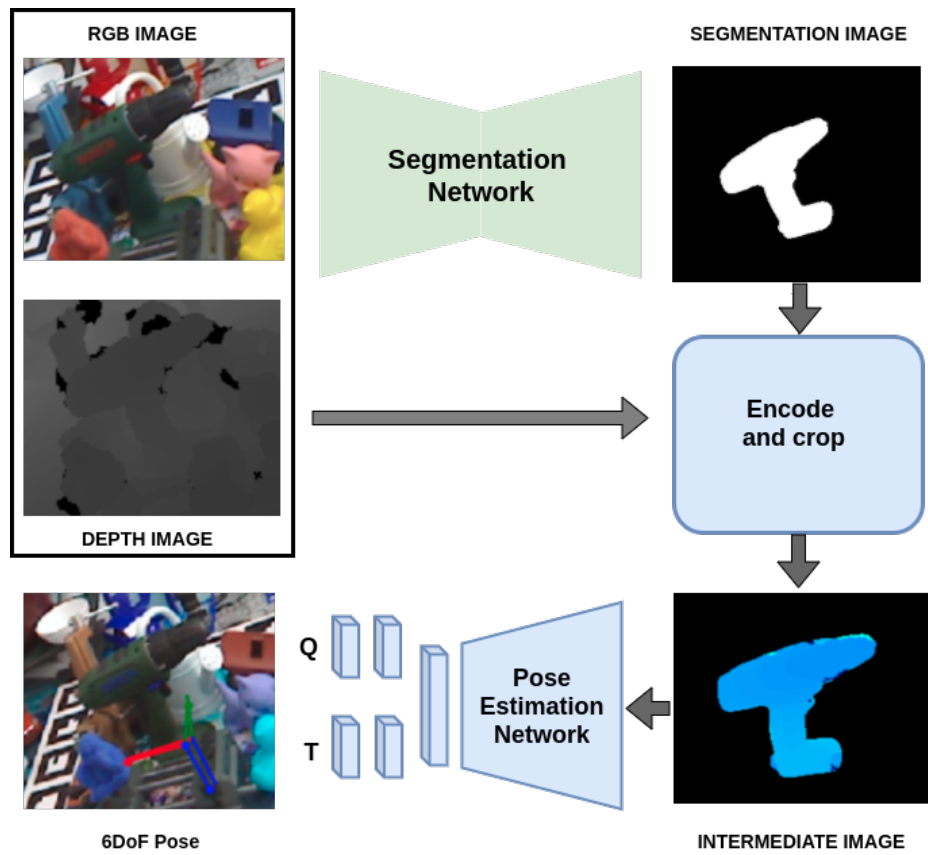


Figure 3.1: Proposed 2-stage pipeline consisting segmentation stage and pose estimation stage. Q in the figure stands for quaternion vector of dimension 4×1 and T stands for translation vector of dimension 3×1 .

3.2 2D Image Segmentation

This section builds the foundation to understand semantic segmentation in 2D images. The architecture of SegNet and the changes made are discussed in the following.

3.2.1 Understanding Image segmentation

Semantic segmentation is an approach to partition image pixels into groups of similar context. Unlike the object detection algorithms which draw a bounding box around an object, semantic segmentation algorithms use pixel-wise labeling strategies to indicate pixels of a similar class or an object. Semantic segmentation is widely used in scene understanding.

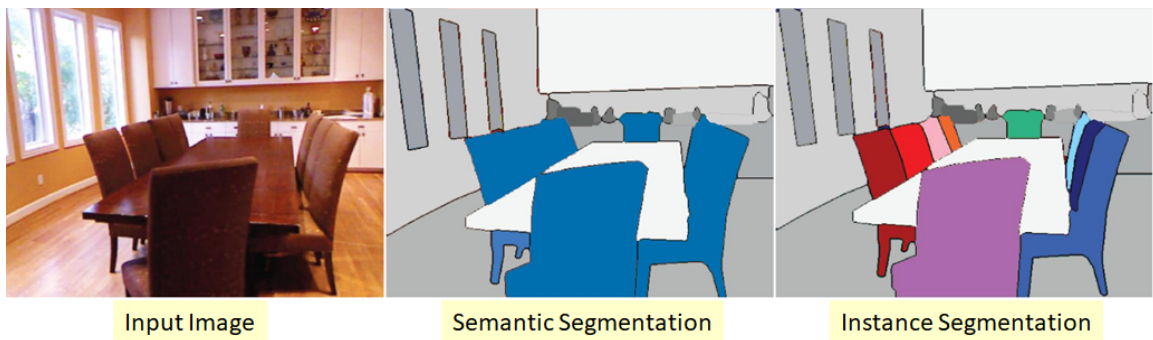


Figure 3.2: A sample image showing semantic segmentation and instance segmentation [38]

Classical computer vision techniques perform semantic segmentation by separating the foreground and background. These methods are mainly categorized as Layer-based and block-based segmentation. The layer based techniques consist of banks of filters in order to define binary masks for the objects, whereas the block-based methods use information from various features of the image to collectively generate segmentation masks. A review paper [41] elaborates these techniques in detail.

With the advancement in deep learning and especially deep convolutional neural networks, there has been a significant improvement in the accuracies as well as inference speed of these segmentation algorithms. A general deep segmentation architecture consists of an encoder and a decoder. This encoder is a set of convolution layers and downsampling layers stacked together in descending order of dimensions. The output of the encoder is a high dimensional representation of the image which only encodes the most significant information.

The decoder, on the other hand, takes this information and tries to reconstruct this image as a binary image. This binary image is the segmentation mask in which the object pixels are distinguished from the background. A more in-depth review of the state of the art deep learning algorithms for semantic segmentation is discussed by [12].

3.2.2 SegNet

SegNet [1] follows an encoder-decoder architecture consisting of multiple layers of convolutional neural networks. The encoder uses the first 13 convolution layers from VGG-16 network. VGG [36] models are deep convolutional neural network architectures which performed significantly well on ImageNet challenge [5]. In this way, the encoder can be initialized with pre-trained VGG16 weights, which gives the network a better starting point in the training process. The last layer of the encoder contains all the high-dimensional information of the image which is later used to reconstruct a segmentation mask.

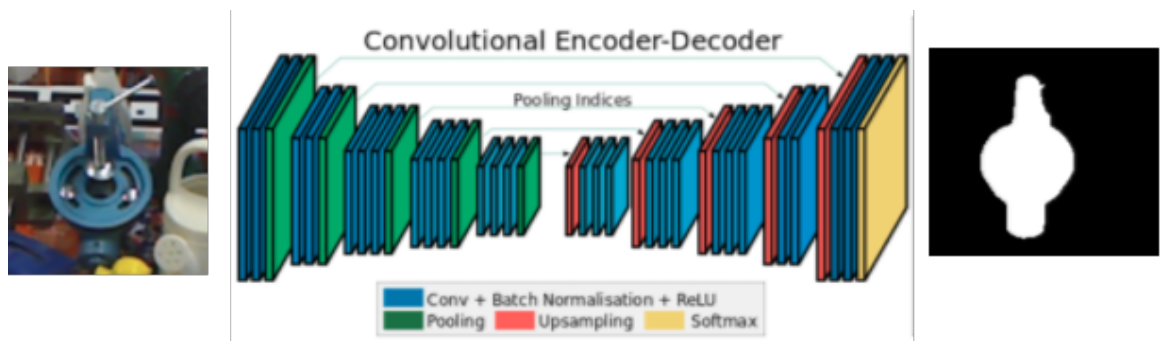


Figure 3.3: Architecture of Segnet [1]

The SegNet differs from other networks by its decoder architecture. The decoder is a mirror image of the encoder except the Pooling layers are replaced by up-sampling. SegNet stores the indices of the values during max-pooling and uses them in the decoder stage to up-sample in the same way as it was down-sampled. This way it saves a lot of memory without much loss of accuracy.

3.2.3 Implementation of 2D segmentation network

As this segmentation block is generic, a Vanilla SegNet architecture with minor changes was used. A dropout layer was introduced after every convolution block in the encoder and decoder stage to avoid over-fitting. A standard dimension of 640x480 was chosen as input to the network with mean - normalized images. The output layer is a single channel probability map with softmax output. This softmax output is converted to a binary segmentation map using a threshold of 0.5. The loss function is a combination of Binary Cross Entropy Loss and a custom loss function which penalized Intersection over Union 3.26.

SegNet was chosen because of its performance and speed. For more accuracy, networks like U-Net [34] and Deep-Lab [4] perform better. For instance segmentation, MASK-RCNN [14] can be used to generate segmentation proposals and pose estimation network can iterate over all the proposals to find the object poses respectively.



Figure 3.4: An RGB Image and a binary mask inferred from Segmentation network

3.3 Object Pose estimation

This section builds the foundation to understand pose estimation. Later, the architecture of the pose estimation network is discussed in detail.

3.3.1 Understanding Object Pose Estimation

Estimating the pose of given objects from the 2D image is generally done by computing a relationship between the 2D image coordinates and the 3D point coordinates. In this approach, the model is projected on an image plane with multiple poses and the projected image is matched with the original to select the pose with least error. Similarly, a 3D approach uses object models and computes a transformation between the scene and the model using registration algorithms like Iterative closest point.

Direct Linear Transform

The Direct Linear transform is the simplest and one of the early algorithms used to estimate the camera pose from 2D image key-points by estimating the homography matrices [7] [28] [29]. For a simplified 2D image the 2D world coordinates of the objects can be converted to u,v co-ordinates using the homography matrix given by 3.1. The x,y,z coordinates are the translations in x,y,z axes with respect to the camera frame. Whereas, the u,v coordinates are location of the pixels on an image plane with respect to the top left corner of the image.

$$c \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (3.1)$$

where H is given by 3.2 and c is a scalar non-zero constant.

$$H = \begin{bmatrix} h1 & h2 & h3 \\ h4 & h5 & h6 \\ h7 & h8 & h9 \end{bmatrix} \quad (3.2)$$

The above equation 3.1 can be written in the form $A \cdot h = 0$, where A is given by stacking 3.3, 3.4 to form A matrix 3.5 and h matrix by 3.6.

$$-h1 \cdot x - h2 \cdot y - h3 + (h7 \cdot x + h8 \cdot y + h9) \cdot u = 0 \quad (3.3)$$

$$-h4 \cdot x - h5 \cdot y - h6 + (h7 \cdot x + h8 \cdot y + h9) \cdot v = 0 \quad (3.4)$$

$$A = \begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & u \cdot x & u \cdot y & u \\ 0 & 0 & 0 & -x & -y & -1 & v \cdot x & v \cdot y & v \end{bmatrix} \quad (3.5)$$

$$h = [h1 \quad h2 \quad h3 \quad h4 \quad h5 \quad h6 \quad h7 \quad h8 \quad h9]^T \quad (3.6)$$

The solution to $A \cdot h = 0$ is usually found out by Single Value Decomposition. It computes the eigen values of covariance and eigen vectors of A and returns solution along principal axis corresponding to minimum singular value [29]. Other optimization techniques based on different cost-functions are described in [7].

Perspective n Points - PnP

For estimating the homography matrix for 3D point, the homography matrix is represented as a combination of camera intrinsic and extrinsic matrix 3.7.

$$c \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R11 & R12 & R13 & T_x \\ R21 & R22 & R23 & T_y \\ R31 & R32 & R33 & T_z \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3.7)$$

When there are n sets of 3D-points with their 2D correspondences and known camera intrinsics, the pose of the object w.r.t to the camera can be found out using Perspective n Points or PnP algorithm. PnP can be implemented as an iterative or non-iterative algorithm with at least 3 correspondence points. EPnp [22] is a variant of PnP which gives a solution in $O(n)$ time complexity and is non-iterative. It represents a point as the weighted sum of 4 virtual non-coplanar control points, explained in equation 3.8. See figure 3.5.

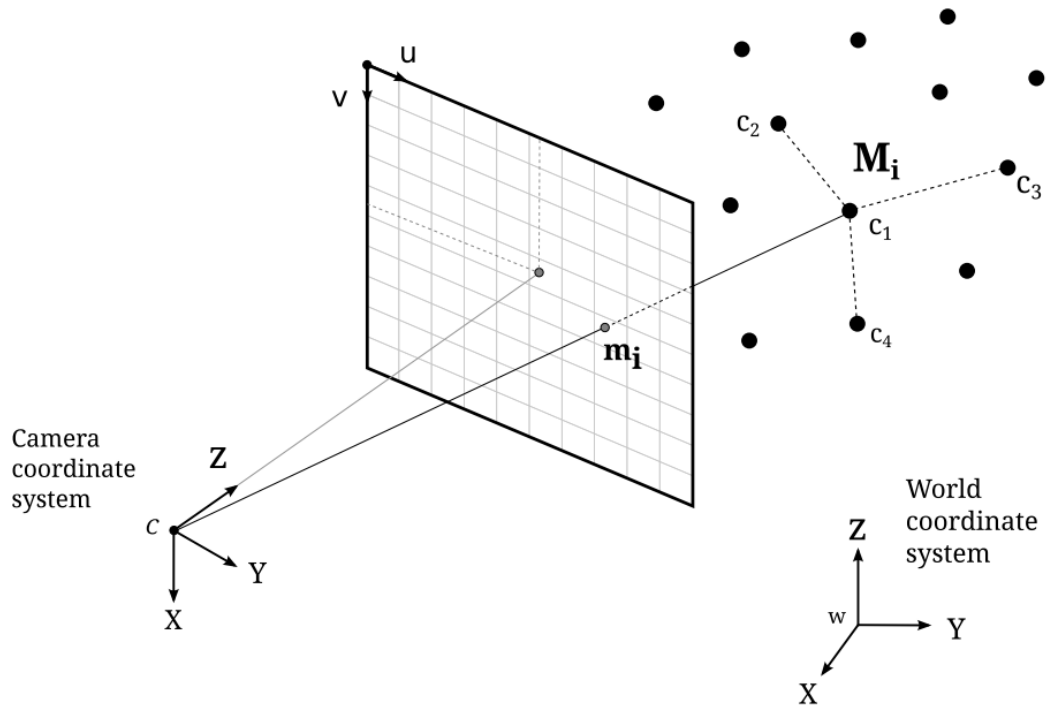


Figure 3.5: Pose estimation using EPnP [29]

$$p_i^w = \sum_{j=1}^4 \alpha_{i,j} \cdot c_j^w \quad (3.8)$$

where $p_i^w = [X^w Y^w Z^w]^T$ is a 3D point in world co-ordinate system and $c_j^w = [X^w Y^w Z^w]^T$ is a 3D control point.

Similarly, using DLT in the control point reference frame, object pose is obtained [29].

Iterative closest Point

Iterative Closest Point is a registration algorithm, which computes a transformation matrix consisting of rotation and a translation of the object with respect to a given model of that object in the sensor frame [26]. The algorithm first computes distinct features in the model point cloud and the scene point cloud. These features are matched using feature matching algorithms to find correspondences.

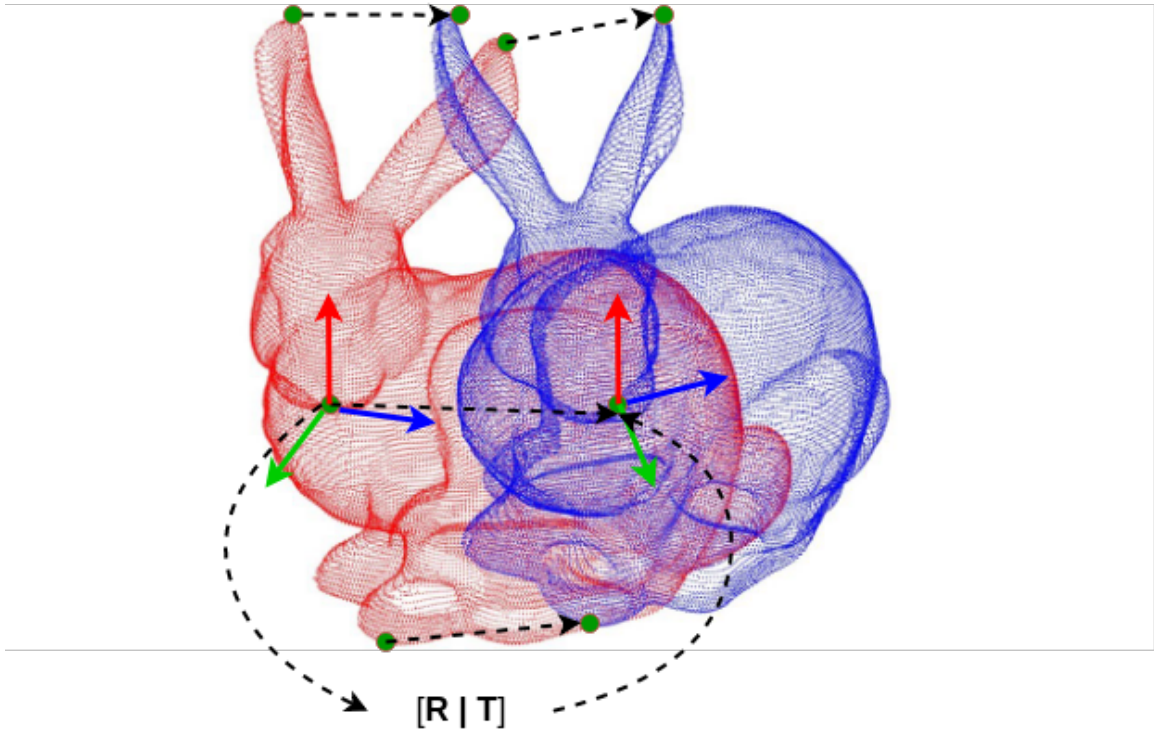


Figure 3.6: Registration of point clouds using Iterative Closest Point [13]

$$d_k = \frac{1}{N} \sum_{i=1}^N \|y_k - (R_k \cdot x_k + T_k)\| \quad (3.9)$$

Where d_k gives the error distance in the sampled point y_k and the modeled point x_k transformed by R_k rotation matrix and T_k translation matrix.

This process is iterated until the convergence criteria is satisfied. Following are the steps:

- compute the correspondences to find y_k and x_k
- compute the registration using equation 3.9 and Singular value decomposition.
- register the model in the scene using the transformations computed
- if $d_k < e_k$ (where e_k is some error threshold), return the transformations else repeat the same for next iteration.

It is generally observed that ICP works best if it is provided with some initial estimate of the object pose, thus making it a good candidate for pose refinement. The coarse pose can be computed by the above mentioned methods like PnP.

3.3.2 Implementation of Pose Estimation Network

The above mentioned methods are good estimators of pose but are very prone to outliers and due to their iterative nature, they are time consuming. In this section, a deep learning architecture is described which learns the transformation of the objects points (represented as depth image) from the camera frame.

A VGG-11 encoder is modified by adding two fully connected regression networks to it. Thus, the network gives two output vectors, a 4x1 quaternion vector for the object pose and a 3x1 vector for location of the center of the object in a 3D co-ordinate space. At every forward pass the quaternion is normalized to make it a unit quaternion.

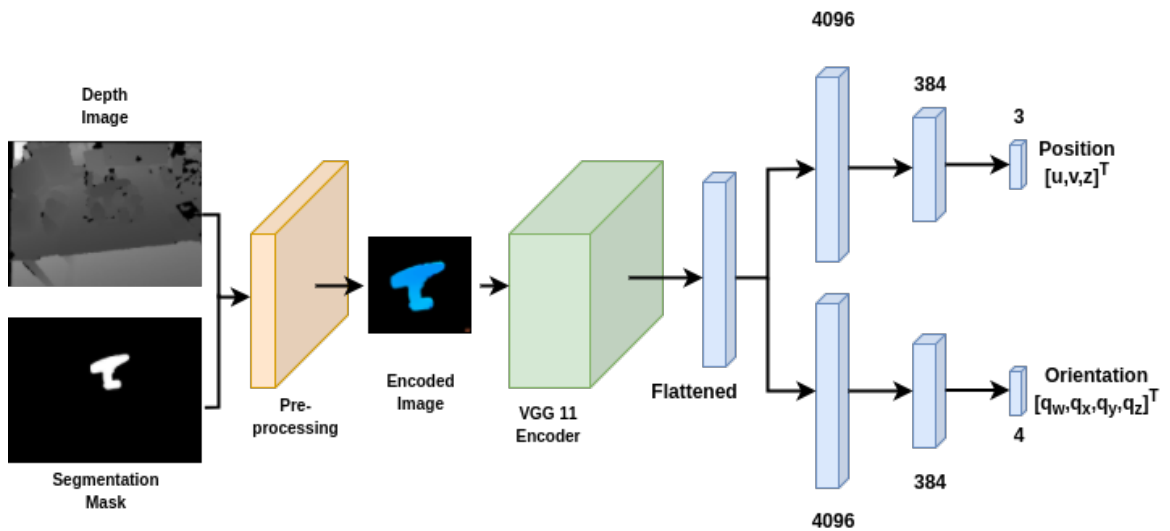


Figure 3.7: Network architecture for object pose estimation

$$\hat{q} = \frac{\hat{q}}{|\hat{q}|} \quad (3.10)$$

The segmentation mask obtained from the previous stage is used to crop out the depth values corresponding to the object. A square bounding-box is then fitted around the cropped depth image and resized to (224 x 224). The resultant image obtained is a square with only the depth values of the corresponding object.

As this is a single channel image, this information is encoded into 3 channels using a JET encoding. JET representation encodes the information in depth image into RGB space

which makes it structurally similar to an RGB image. The depth values in the depth image are normalized and are up-sampled to form a 3 channel 8 bit image. The points with lesser depth value are green in color and the values which are farther are red. See figure 3.8

JET encoding is proven to be helpful in object detection tasks with depth as additional data [8]. As the VGG11 encoder was pre-trained on ImageNet dataset, transfer learning with JET encoded depth images do not change the weights drastically.

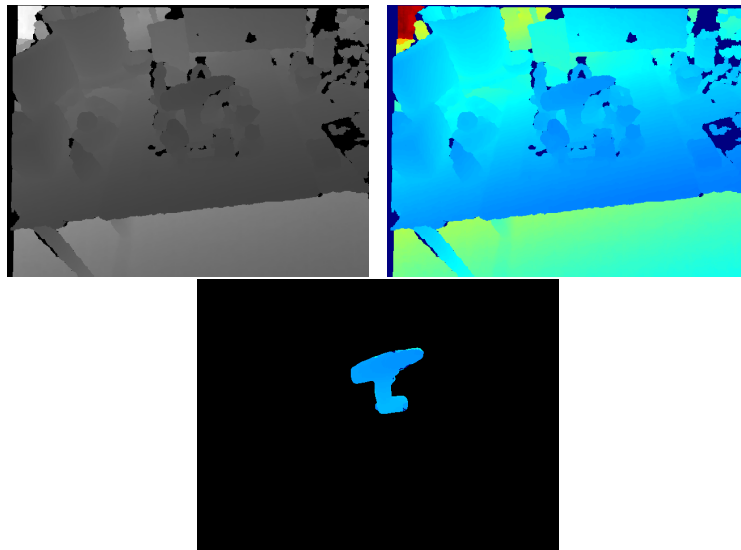


Figure 3.8: Depth image (top left), its JET encoding representation (top right) and image after element-wise multiplication (bottom)

Pose estimation loss function is defined in [39] which estimates the loss as the Euclidean distance between sample points of the object which are transformed by predicted pose and its corresponding ground truth pose. Details of the loss function are discussed in 3.4.2

To infer the position of the object the position regression fully connected branch of the network regresses a 3×1 vector. As addressed by [40], the naive approach of regressing pose as 3D coordinates (x,y,z) of the center of the object cannot be generalized and is prone to errors. It would also depend on the image dimensions and camera intrinsics. To overcome this problem the position of the object center is represented in (u,v,z) coordinate representation.

$$x = \frac{(u - c_x) \cdot z}{f_x} \quad (3.11)$$

$$y = \frac{(v - c_y) \cdot z}{f_y} \quad (3.12)$$

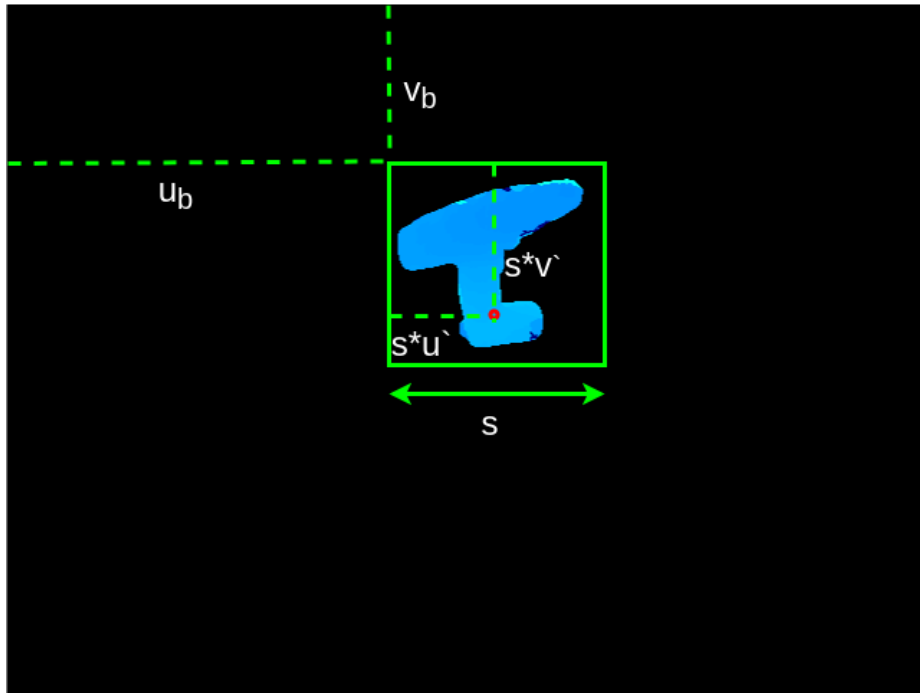


Figure 3.9: Representation of object center in u,v coordinate space

The top left co-ordinates of the bounding box were stored (u_b, v_v) before cropping the depth image. The network regresses the center of the object (u', v', z) , where u' and v' are normalized co-ordinates of the object center with respect to the bounding-box.

$$u = u_b + s \cdot u' \quad (3.13)$$

$$v = v_b + s \cdot v' \quad (3.14)$$

where s is the side of the bounding box. u' and v' are normalized distance and can change from 0.0 to 1.0.

The rotation matrix is computed from the quaternion using the equation 3.15 and similarly translation matrix from the equation 3.16. These mathematical operations are implemented as tensors using pyTorch which gives the ability to perform these computations on a GPU.

$$R = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x \cdot q_y - q_z \cdot q_w) & 2(q_x \cdot q_z + q_y \cdot q_w) \\ 2(q_x \cdot q_y + q_z \cdot q_w) & 1 - 2(q_x^2 + q_z^2) & 2(q_y \cdot q_z - q_x \cdot q_w) \\ 2(q_x \cdot q_z - q_y \cdot q_w) & 2(q_y \cdot q_z + q_x \cdot q_w) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix} \quad (3.15)$$

$$T = \begin{bmatrix} x & y & z \end{bmatrix}^T \quad (3.16)$$

3.4 Loss Function

This section discusses about the Loss functions in detail used in the segmentation network as well as in the pose estimation network.

3.4.1 2D segmentation loss

Binary Cross Entropy

The segmentation map generated by the segmentation network is a probability map of pixels belonging to the object. Thus, in a perfectly inferred segmentation map, the pixels belonging to the object will have a value close to 1.0 and the pixel belonging to the background will have values close to 0.0. Binary Cross Entropy Loss (BCE) computes the cross-entropy between the ground truth label distribution and the predicted label probability distribution. As the raw neural network values can be outside the range of 0.0-1.0, the output is normalized using a sigmoid function. 3.17

$$p_i = \frac{\exp e^{s_i}}{\sum_j^N \exp e^{s_j}} \quad (3.17)$$

Here p_i is the softmax output of i^{th} pixel and s_i is the raw inferred value of the network.

$$BCE = -\frac{1}{N} \sum_{i=0}^N (y_i \log(\hat{y}_i) + (1 - y_i) \log((1 - \hat{y}_i))) \quad (3.18)$$

Where N is the number of pixels in an $m \times n$ image, $N = m \times n$, y is the ground truth label and \hat{y} is the predicted softmax value of that pixel.

Example: Computing Binary Cross Entropy

Consider an image of dimension 3×3 . The ground truth labels corresponding to the object is given by 3.19.

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad (3.19)$$

Similarly a predicted map is obtained from the Segmentation Network given by 3.20 which represents the probabilities with which the pixel belongs to the object.

$$\begin{bmatrix} 0.12 & 0.90 & 0.76 \\ 0.09 & 0.89 & 1.00 \\ 0.22 & 0.54 & 0.61 \end{bmatrix} \quad (3.20)$$

The Binary Cross Entropy loss is computed between the ground truth 3.19 and 3.20 by the equation 3.18.

For $i = 0$,

$$BCE_0 = -1 \cdot (0 \cdot \log(0.12) + (1 - 0) \cdot \log(1 - 0.12)) = 0.127$$

Accordingly computing BCE for $i=0$ to $i=8$ and then summing them up, $BCE = -2.695$ is obtained.

Intersection over Union

Intersection over Union (IoU) is a metric to evaluate the overlap of ground truth and predicted segmentation masks. As the overlap of the predictions over ground truth needs to be maximized, a function of IoU as loss function is used.

$$IoU = \frac{|A \cap B|}{|A \cup B|} \quad (3.21)$$

Example: Intersection over union After a binary threshold operation with threshold value of $\tau = 0.5$, a binary label map is obtained given by 3.22

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad (3.22)$$

For the ground truth 3.19 and predicted mask 3.22 the intersection is given as 3.23 and Union is given as 3.24

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad (3.23)$$

$$\begin{bmatrix} 0 & 2 & 2 \\ 0 & 2 & 2 \\ 0 & 1 & 1 \end{bmatrix} \quad (3.24)$$

IoU is computed as

$$IoU = \frac{Intersection}{Union} = \frac{4}{10} = 0.4 \quad (3.25)$$

2D Segmentation Loss function

The final loss function used for training the segmentation network is given by 3.26 which incorporates both binary cross entropy loss and Intersection over union as a weighted sum. As IoU varies from 0.0 \rightarrow 1.0 $-\log(IoU)$ changes from $\infty \rightarrow$ 0.0 respectively. Thus, lower the IoU, higher is the loss.

$$L_{segmentation} = BCE - \alpha \cdot \log(IoU) \quad (3.26)$$

3.4.2 Pose Estimation Loss

To train the Pose estimation network, a loss function defined in [39] is used, which tries to minimize the euclidean distance between ground truth and predicted point on the object.

$$L_{pose} = \frac{1}{M} \sum_{j=0}^M \|(Rx_j + t) - (\hat{R}\hat{x}_j + \hat{t})\| \quad (3.27)$$

A set of M points are randomly sampled from the object (x_j) and transformed with the ground truth transformation matrix given by $[R|T]$. Similarly the same points are transformed with their corresponding transformation predicted by the network. Here the transformation $[\hat{R}|\hat{T}]$ is obtained by converting the (u,v,z) co-ordinates to world co-ordinates using 3.13 , 3.14 , 3.15 and 3.16.

Chapter 4

Experimentation and Results

4.1 Experimentation and Evaluation on LINEMOD dataset

4.1.1 LINEMOD Dataset

Linemod [17] dataset is commonly used to benchmark object pose estimation algorithms. It consists of around 18000 images and ground truth pose for 15 different objects. The RGB images and corresponding depth images were captured using the Kinect sensor under indoor lighting conditions. All images are resized to 640x480 and are stored in jpg format. The dataset consists of the following files

- RGB images - 640x480 saved in jpg format
- distance.txt - contains maximum diameter of object in cm.
- transform.dat - contains the ground truth pose of the object as an SO4 transformation.
- OLDmesh.ply - object mesh
- mesh.ply - object mesh after registration.

As the Linemod dataset does not provide pixel-wise segmentation masks, the object points from the 'mesh.ply' files which were transformed according to the ground-truth pose were projected onto the images plane to generate a binary mask.

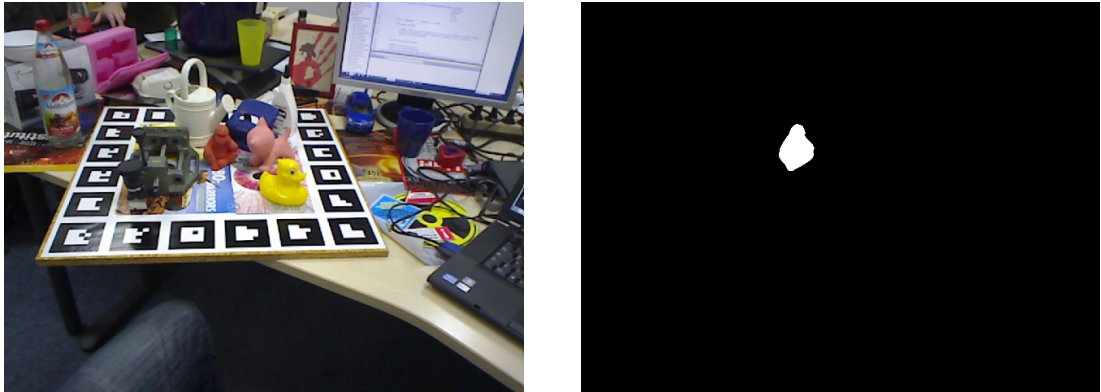


Figure 4.1: Image of Ape from Linemod dataset (left), corresponding segmentation masks(right)

4.1.2 Data augmentation

Deep neural networks often require large samples of the dataset to train. It is difficult to generate new image samples from the same image distribution. To overcome this problem image augmentation is used to generate more sample of images from the same distribution of images. A library albumentation [2] was heavily used to augment images to generate fake images. Following are the augmentations used in the training pipeline in an inline manner. The probability of the augmentation signifies that the chances of it being picked as an augmentation while pre-processing an image.

- One of (RandomContrast, MedianBlur, RandomBrightness) - probability(0.5)
- VerticalFlip - probability(0.5)
- HorizontalFlip - probability(0.5)
- ShiftScaleRotate - probability(0.5)

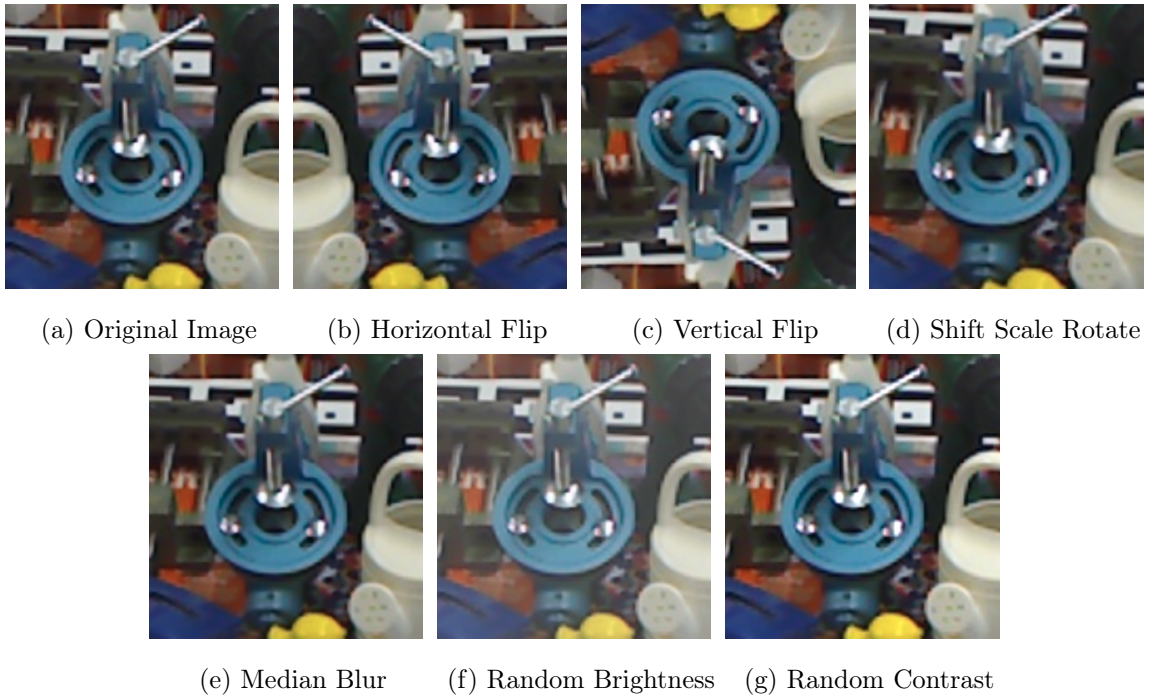


Figure 4.2: Sample image and its augmentations used during training

4.1.3 Training

Both the convolutional networks are implemented in PyTorch[25] and used GPU for training. The segmentation network was trained from scratch using Adam optimizer with a learning rate of $1e - 4$ and for 200 epochs. To avoid over-fitting it was empirically found out that a dropout rate of 0.1 worked well. The loss function used to optimize this network is defined in 3.26. Similarly, the pose estimation network uses Stochastic Gradient Descent optimizer and a learning rate of $1e - 4$. The network was initialized with pre-trained weights trained on ImageNet dataset provided by PyTorch framework. The loss function used to optimize this network is defined in 3.27.

4.1.4 Segmentation Results

The accuracy of the segmentation network is evaluated with Jaccard Index, also known as Intersection over Union.

$$IoU = \frac{|p_{bin} \cap g|}{|p_{bin} \cup g|}$$

where p_{bin} is the binary value of the predicted mask after performing a binary threshold on the sigmoid output at 0.5. $p_{bin} \in \{0, 1\}$. g is the binary ground truth mask

Table 4.1: Segmentation Performance on LINEMOD Images

Objects	IoU
ape	0.944
benchvise	0.931
cam	0.929
can	0.938
cat	0.922
driller	0.928
duck	0.923
eggbox	0.931
glue	0.929
holepuncher	0.942
iron	0.931
lamp	0.922

Segmentation Performance on LINEMOD Images

Table 4.2: Statistics of Segmentation Performance on LINEMOD Images

Mean	STD
0.9308	0.006

Average segmentation performance on LINEMOD Images

Table 4.1 shows IoU scores for all the 12 categories. The network shows an average IoU score of 0.9308 and a very low standard deviation which indicates robustness of the performance, refer table 4.2.



Figure 4.3: RGB images of ape, drill and bench-wise (left), their corresponding masks (Right).

4.1.5 Pose Estimation Results

To evaluate the accuracy of the final pose regressed by the Pose estimation network, Average distance metric is used proposed by [17]. A random point is sampled from the object and transformed with ground truth pose to obtain x_g . The same point is then transformed with the predicted pose to obtain a point x_p . The error in the ground truth pose and predicted pose is given by the Euclidean distance between x_g and x_p . refer 4.1.

$$d = \frac{1}{M} \sum_{j=0}^M \|(Rx_j + T) - (\hat{R}\hat{x}_j + \hat{T})\| \quad (4.1)$$

For a given threshold τ (in cms). If the distance is less then a given threshold, The pose is accepted else the pose is rejected 4.2.

$$f = \begin{cases} 1 & d \leq \tau \\ 0 & otherwise \end{cases} \quad (4.2)$$

The pose which is accepted is considered as a hit and the pose which is rejected is considered as miss. The accuracy of the model is computed by 4.3

$$accuracy = \frac{hit}{(hit + miss)} \quad (4.3)$$

Figure 4.4 shows the ground truth pose and predicted pose of samples in the test set. The axes with a blob at the tip represents ground truth pose, whereas the other represents the predicted pose. These poses are projected on the original image for visually understand the results.

The pose estimation network was evaluated on varying distance thresholds for its accuracy response. Figures 4.5, 4.6 shows the trend of accuracy vs the distance threshold. It can be observed that the distance accuracies are more than 90% when the threshold is greater than 30% of the object diameter. Moreover, the error values are large, this is due to its symmetry. There can be multiple valid pose for symmetric objects.

Comparing with BB8 [32], the pose accuracy of BB8 is more than 90% for threshold as 20% of the object diameter. The proposed approach does not perform well against BB8 as only depth information is used and no RGB features are learnt.

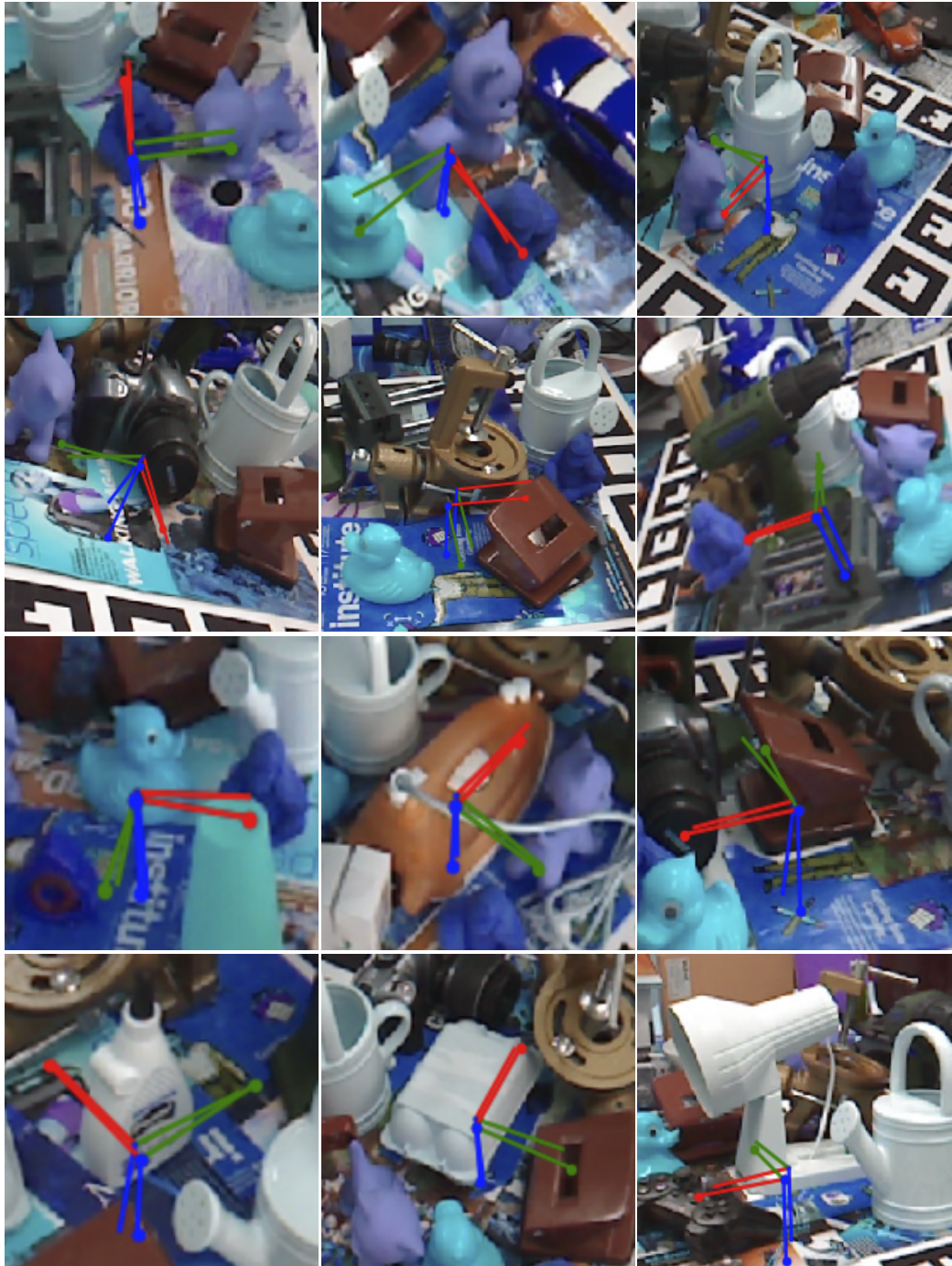


Figure 4.4: Predicted and ground truth pose overlaid on the RGB images. The axes with a blob at the tip is the ground truth pose, the other is the predicted pose

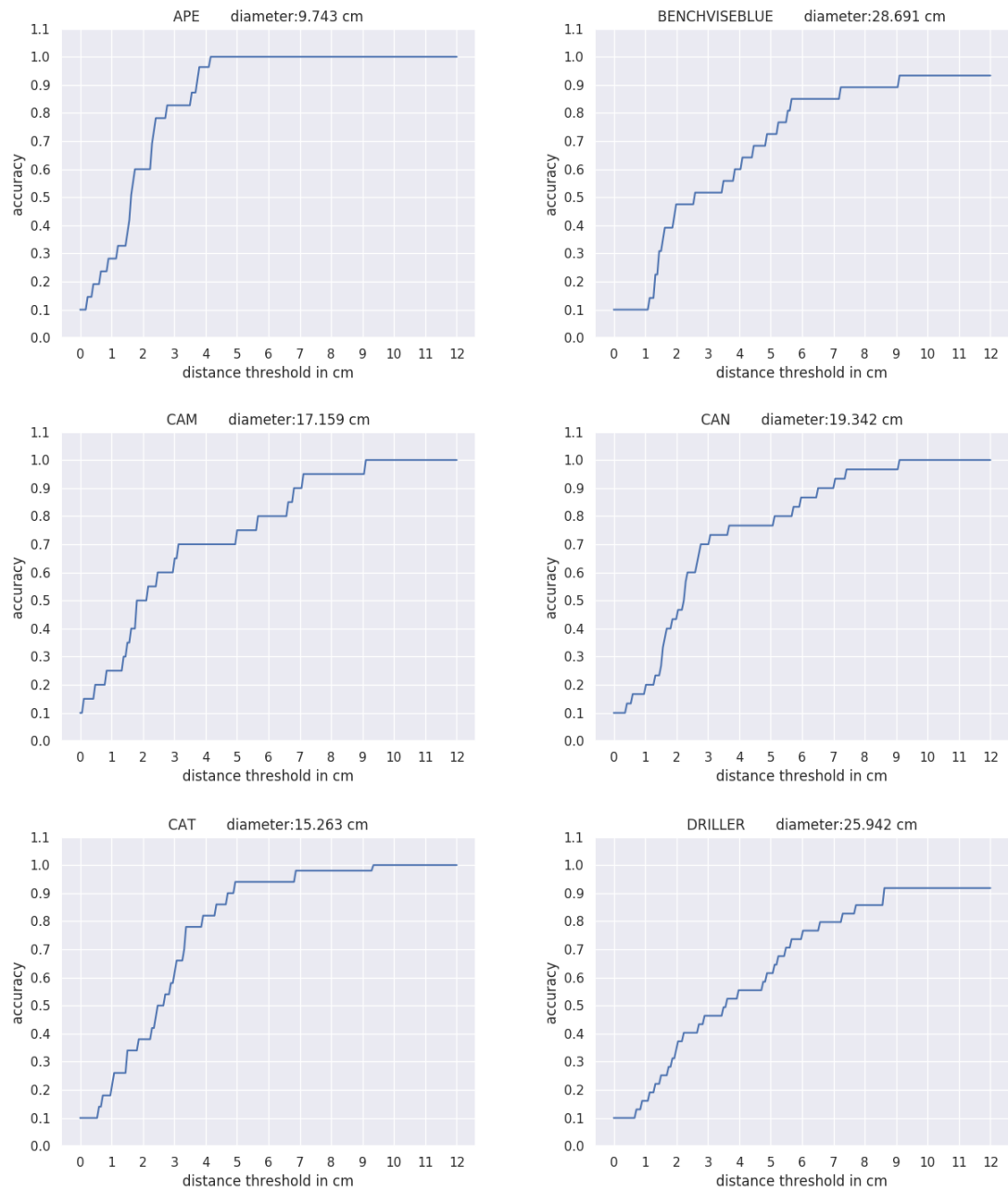


Figure 4.5: Trends of accuracy with change in distance threshold for (from top left to bottom right) ape, benchwise, cam, can, cat, drill

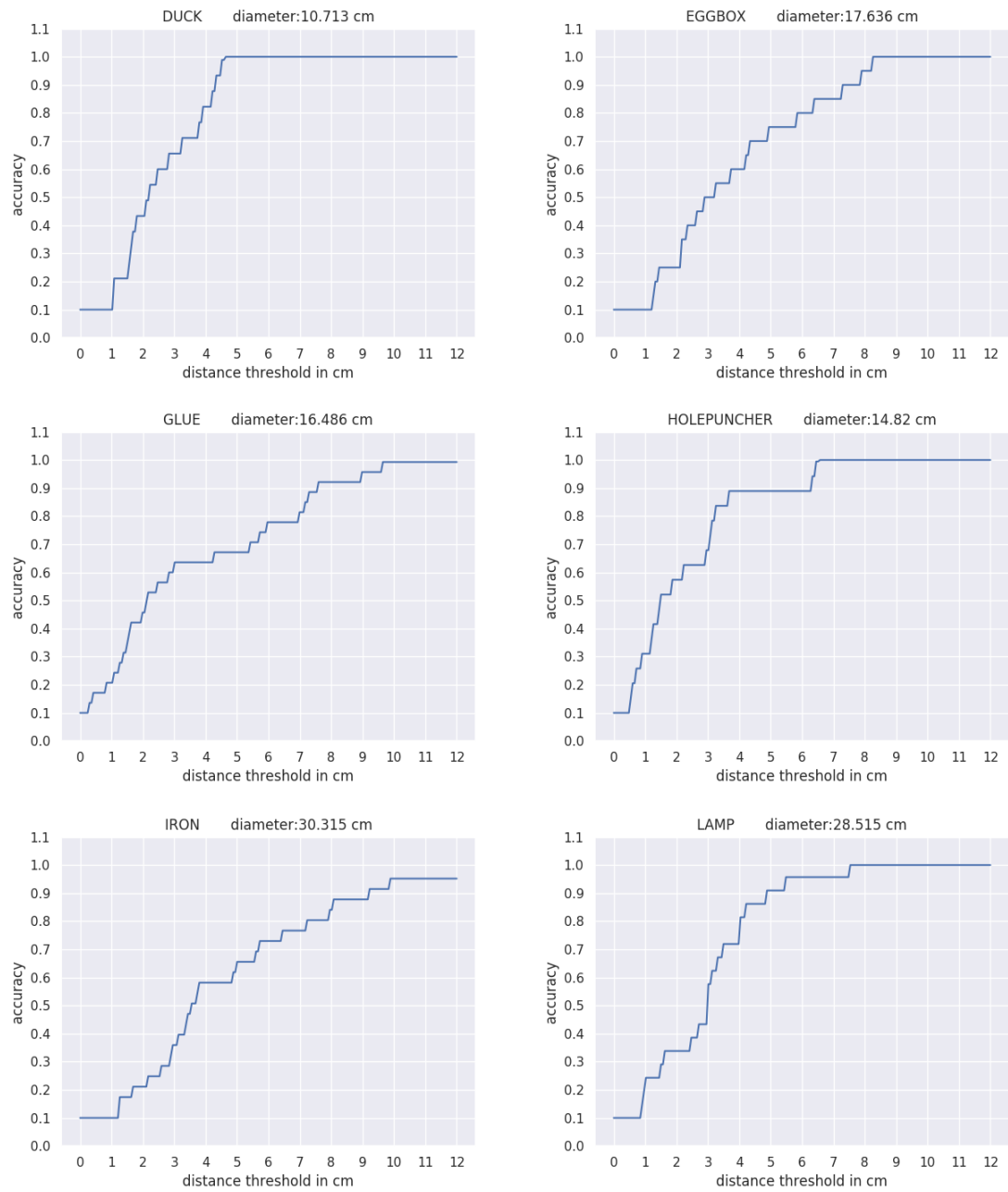


Figure 4.6: Trends of accuracy with change in distance threshold for (from top left to bottom right) duck, egg-box, glue, hole-punch, iron, lamp

Inference time

Table 4.3 shows the split of inference time of individual stages. The framework takes an average of 22.7 milliseconds for a forward pass. This is due to the smaller network size and light-weight architectures. Both the networks fit on one GTX 1060Ti GPU on which the performance was measured.

Table 4.3: Inference Speed on LINEMOD dataset

Stage	Time
Segmentation	22.1 msec
Pose Estimation	5.7 msec
Total	27.8 msec

Average speed in milli-seconds for segmentation and pose estimation stage

4.2 Experimentation and Evaluation on ATLAS Robot

4.2.1 Acquiring Real World Images

The data distribution of RGB images of LINEMOD dataset is not similar to the images captured from the Multisense SL[33], refer table 4.4. This is because the camera intrinsic and the camera response curves of Multisense SL are different from that of Kinect sensor. Thus a network trained on LINEMOD images fails to perform well on images captured from other sensors.

To overcome this problem a custom segmentation dataset was created. Around 40 RGB images were captured using the multisense SL sensor with a drill machine as the object of interest, placed at different positions in the scene, refer figure 4.7. These 40 images with the object were annotated for pixel-wise segmentation masks. Refer figure 4.8.

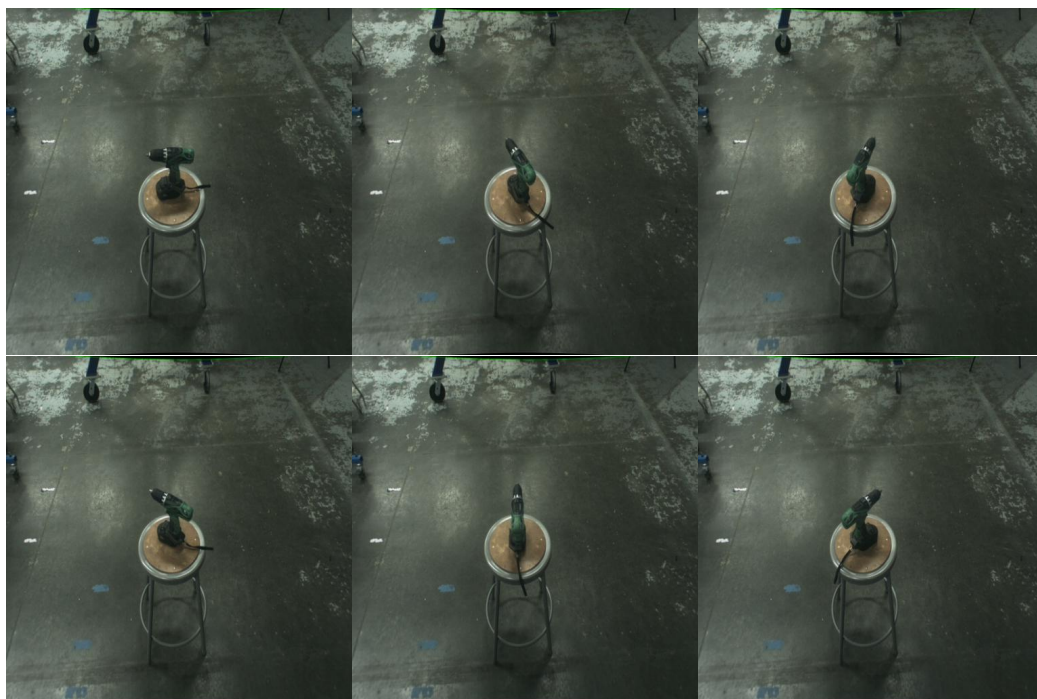


Figure 4.7: Images of the object captured with MultisenseSL

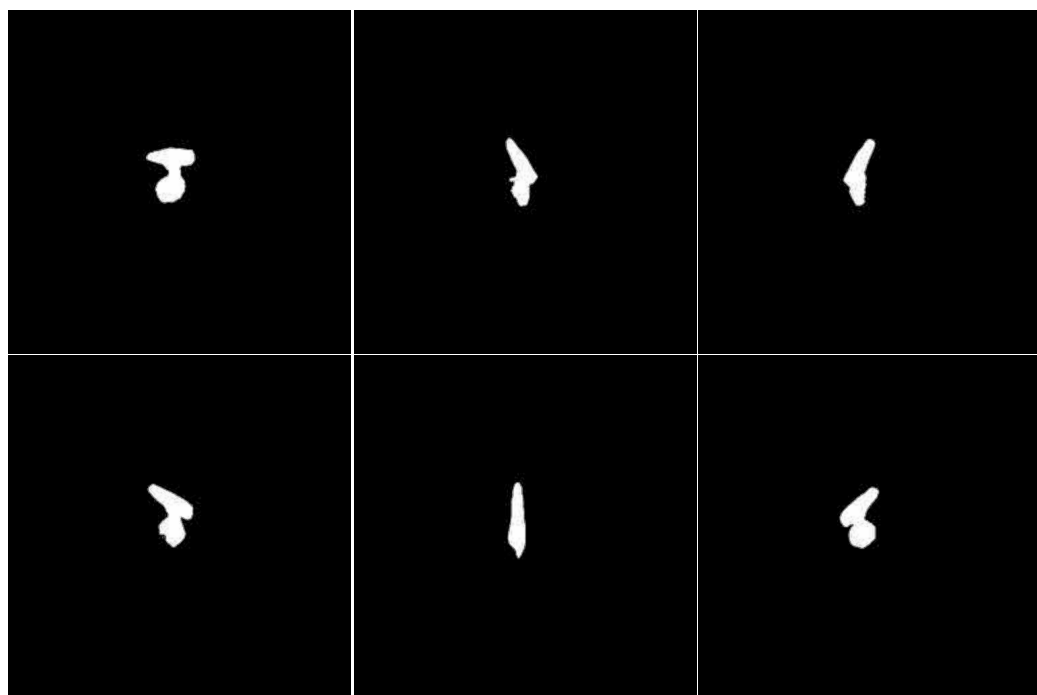


Figure 4.8: Ground truth masks for the real object images

Similarly, around 40 images of the background were captured by moving the sensor randomly, refer figure 4.9



Figure 4.9: Images of the Background captured with MultisenseSL

A python script generated a dataset of 4500 image samples by cropping the objects using the ground truth segmentation marks and randomly placing it on a background image which also were chosen randomly. The above mentioned data augmentations (refer 4.1.2) were also performed online to generate more samples from the distribution.



Figure 4.10: Synthetically Generated Images

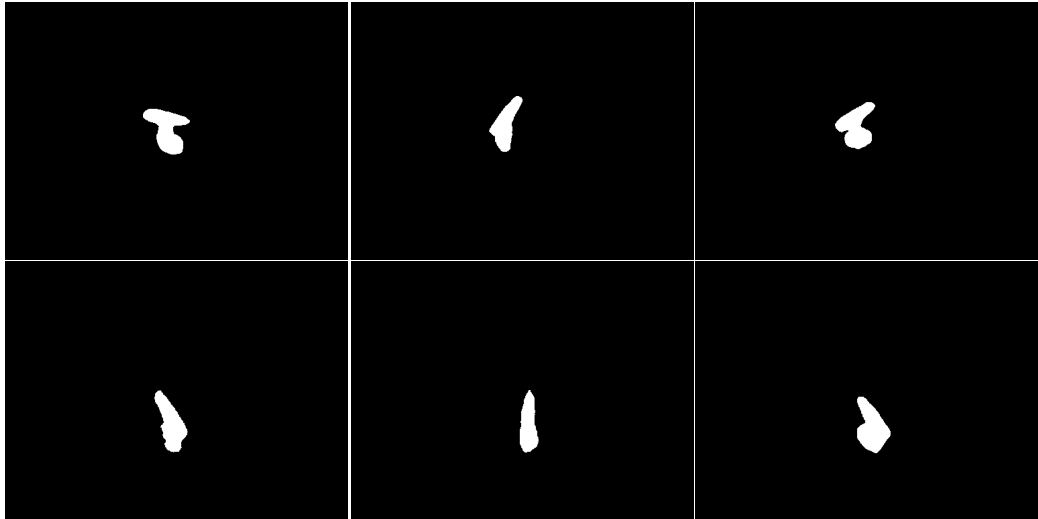


Figure 4.11: Masks for Synthetically Generated Data

Table 4.4: Comparative Analysis of dataset distributions

Dataset	Mean R	Mean G	Mean B	STD R	STD G	STD B
LINEMOD IMAGES	0.333	0.333	0.332	0.191	0.191	0.190
REAL IMAGES	0.181	0.203	0.241	0.213	0.211	0.214

Mean and standard deviations of the R,G and B channels of normalized images in LINEMOD and REAL image dataset

Table 4.4 clearly shows the difference in the distribution of pixel values in R, G and B. LINEMOD dataset is captured in a very controlled lighting condition and can be seen from the standard deviations of the 3 channels, whereas the standard deviation of the images captured from the MultiSense sensor is higher. It can be observed that the mean RED value is significantly less than Green and Blue, which can be observed in the images in 4.7 and 4.9.

Table 4.5: Segmentation Performance on Real world Images

Fold	IoU
0	0.944
1	0.932
2	0.948
3	0.936
4	0.929

Segmentation Performance on Real world Images

Table 4.6: Statistics of Segmentation Performance on Real world Images

Mean	STD
0.9378	0.0071

Average segmentation performance on real world images

To train the segmentation network this dataset was split into 5 folds each of 80% training samples and 20% validation samples. The training procedure remained same as mentioned in 4.1.3. The table 4.5 and 4.6 shows no signs of over fitting as the IoU scores are consistent across all the folds.

4.2.2 Acquiring Depth Images

The primary sensor of ATLAS Robot is MultiSense SL [33]. It consists of a stereo camera pair and a spinning Hokuyo lidar, refer figure 4.12. Thus, there are 2 ways to obtain depth images, one from Hokuyo lidar scans and another from stereo image disparity. A point cloud fetched from a laser sensor can be projected on the image plane, the distance of the point from the plane is used as the value corresponding to the pixels of the depth image. Similarly, disparity images can be converted to depth images using the equation 4.4

$$depth = baseline * focal / disparity \quad (4.4)$$

The point cloud generated by the Lidar takes 1 rotation per 2.5 seconds to update. This is very slow and distorts the point cloud if the object is moved during the laser scan. On the other hand, the point cloud generated from the disparity image is observed to be noisy. As the pose estimation is not affected significantly by occlusions and noise, the depth maps are generated using disparity images obtained from the sensor.

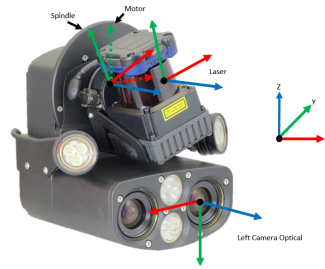


Figure 4.12: MultiSense SL sensor used on the ATLAS robot [33]

4.2.3 TOUGH

The Atlas robot is configured to work with IHMC Controllers (version 0.11) which use ROS kinetic as an interface. TOUGH APIs [9] is a high-level C++ API library to control the robot. It was developed at WPI Humanoid Robotics Lab to serve as a research framework for fast prototyping of software and algorithms on humanoid robots supported by IHMC controllers. This framework was tested on robots such as Valkyrie and ATLAS in simulation as well as on Physical robots. The API framework consists of multiple packages for controllers, motion planners, perception and navigation, refer 4.13. A graphical user interface is provided to quickly control the robot manually or monitor its states.

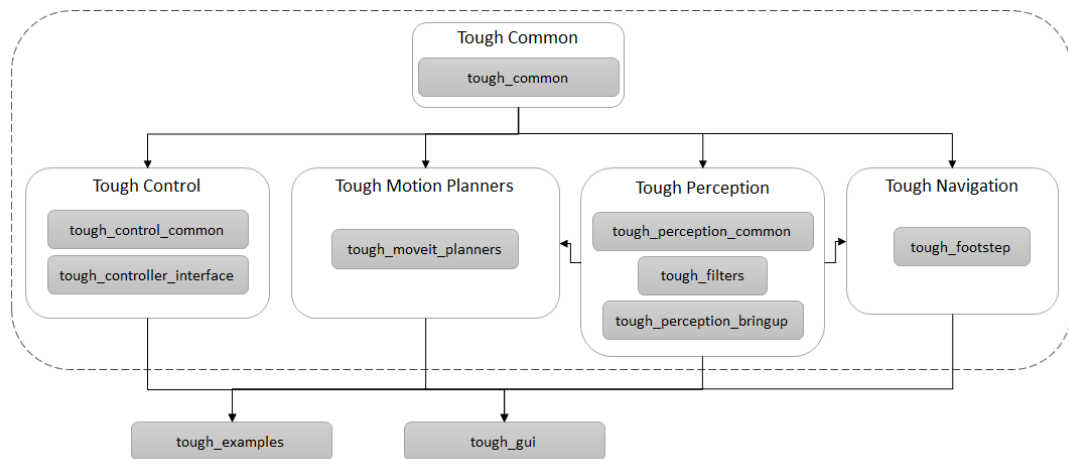


Figure 4.13: TOUGH API overview [9]

The scope of this research uses TOUGH Motion planning package for planning and execution of trajectories in the object pick-up experiment.

4.2.4 ROS nodes

To use the object detection and pose estimation pipeline with the ATLAS Robot, ROS-kinetic [31] is used as a platform to communicate with the Robot. Two ROS nodes are created, one to detect the object and one to manipulate and grab the object. The object detector node consists of a subscriber which listens to the image and depth message topics from multisense SL directly and pre-processes images into RGB and the depth image formats

respectively. For this particular task RGB and depth, images need to be synchronized and are done by using *ApproximateTimeSynchronizer* class of "message filter" package which is available in ROS. The subscriber callback updates a global placeholder for RGB image and Depth depth and race conditions are avoided using Mutex locks.

Algorithm 1 Drill Detection node

```

1: procedure IMAGE_DEPTH_SYNC_SUBSCRIBER(image_msg, depth_msg) ▷
   RGB image and depth images are synchronized and stored
2:   RGB_image ← decode(image_msg)
3:   Depth_image ← decode(depth_msg)
4: procedure ROS_LOOP() ▷ ROS node continuously running
5:   while ROS ≠ shutdown do
6:     segmentation_mask ← SegmentationModel(RGB_image)
7:     object_pose ← PoseEstimationModel(Depth_image, segmentation_mask)
8:     object_pose ← MovingAverageFilter(object_pose)
9:     broadcastObjectPose(object_pose)

```

The ROS Loop implements the main algorithm of object detection and pose estimation. Both the segmentation and pose estimation network models run on GPU and need multi-threaded workers to load and fetch images from the GPU. This is handled by IO functions in PyTorch Library. First, the RGB image is passed to the Segmentation model to generate a pixel-wise binary mask. The Pose Estimation Model uses the binary mask and the corresponding depth image to infer the object pose. As the output of the neural networks are not stable and have significant noise, the object pose is seen to jitter. This adds the problem when planning trajectories with TOUGH. To filter the noise in the detected object pose, A Moving average filter is used with a window size of 15. This window size was found out empirically and has less latency and noise.

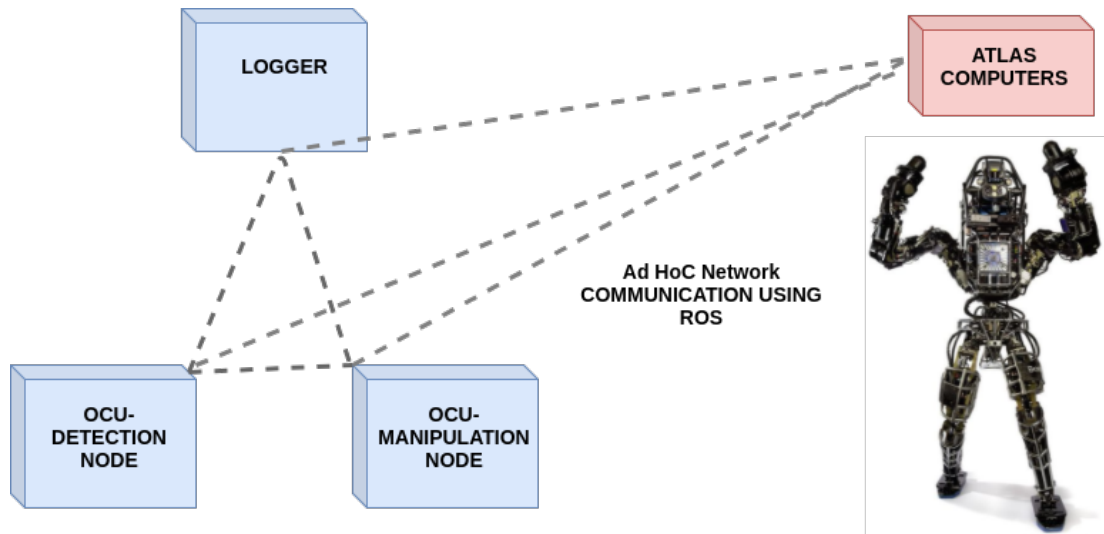
Algorithm 2 Drill Manipulation node

```

1: procedure OBJECT_POSE_SUBSCRIBER(object_pose_msg)      ▷ Listen to
   object pose being broadcasted
2:   object_pose ← decode(object_pose_msg)
3: procedure ROS_LOOP()                                     ▷ ROS node continuously running
4:   while ROS ≠ shutdown do
5:     waitForUser()
6:     TOUGH.planTrajectory(object_pose)
7:     visualizeTrajectory()
8:     approval ← waitForUserApproval()
9:     if approval then
10:      TOUGH.executeTrajectory()

```

Drill Manipulation Node is fairly simple and leverages the high-level APIs provided by TOUGH. A Pose subscriber listens to the objects pose being broadcasted asynchronously. ROS Loop is implemented in an interactive manner where at every stage human approves the action. TOUGH runs Move-it planners (cite moveit) which converts the task space trajectories into joint space trajectories. The *executeTrajectory* procedure converts the trajectory points into the appropriate message and send it to the robot for execution.



OCU - OPERATOR CONTROL UNIT (COMPUTERS)

Figure 4.14: Drill detection on Atlas Robot

Figure 4.14 shows the network architecture to communicate with the ATLAS Robot. A separate gigabit Ad-HoC network is created to have high bandwidth while communicating with the robot. Only the operator control units and 3 onboard ATLAS computer can communicate over this network. As the network is not wireless, the possibility of the connection drops during the experimentation is eliminated. The ATLAS robot runs the ROS master node and IHMC controller on the onboard computers and listens to high levels commands by the TOUGH APIs running on other OCUs. The object detection node runs on an OCU with a GTX-Titan (2012 model with 6GB memory) whereas manipulation node is running on another OCU with the operator in the loop. The logger is used to log all the data being sent to or received from the robot which is then used to analyze the experiment later in time.

4.2.5 Drill Detector Task

To demonstrate the functioning of the algorithm on Atlas robot, a drill is kept in front of the Robot and at a reachable location. The drill detection node listens to the RGB images and depth images being published by the MultiSense sensor on the robot and estimates a 6DoF pose. This pose is published as a ROS Pose message. A ROS message of transform is also published for the sake of visualization. Fig 4.15 shows a view of RViz, a visualization software for ROS. The left image in the RViz shows the colored point cloud and the pose of the drill being detected in the scene, whereas the right image shows the RGB image from the left camera and an overlaid segmentation mask.

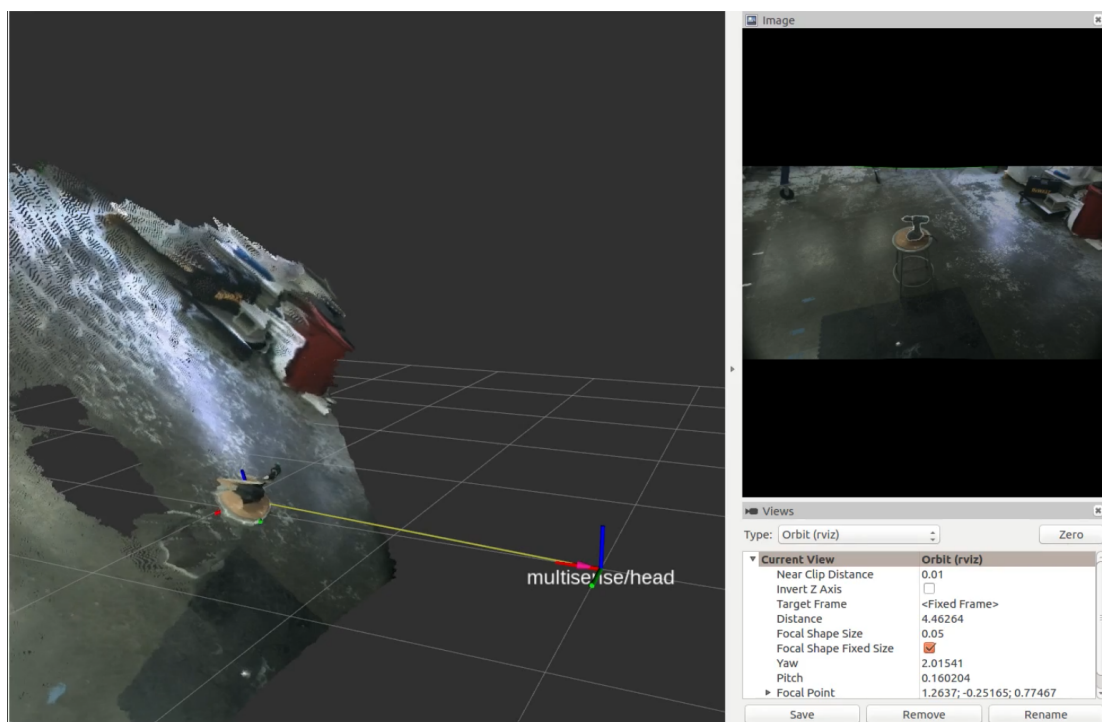


Figure 4.15: Drill detection on Atlas Robot

To verify the performance of the system visually on a moving object, the object was moved randomly in the field of view of the robot. Fig 4.16 shows a sequence of frames of a video. The left image in each frame is the 3D colored point cloud and the image on the right is the RGB image with the segmentation mask overlaid on top. It can be noted from the figure that the segmentation mask sometimes detects random objects as the drill or multiple

proposals. To fix this issue, only the proposal which has the largest area is considered.

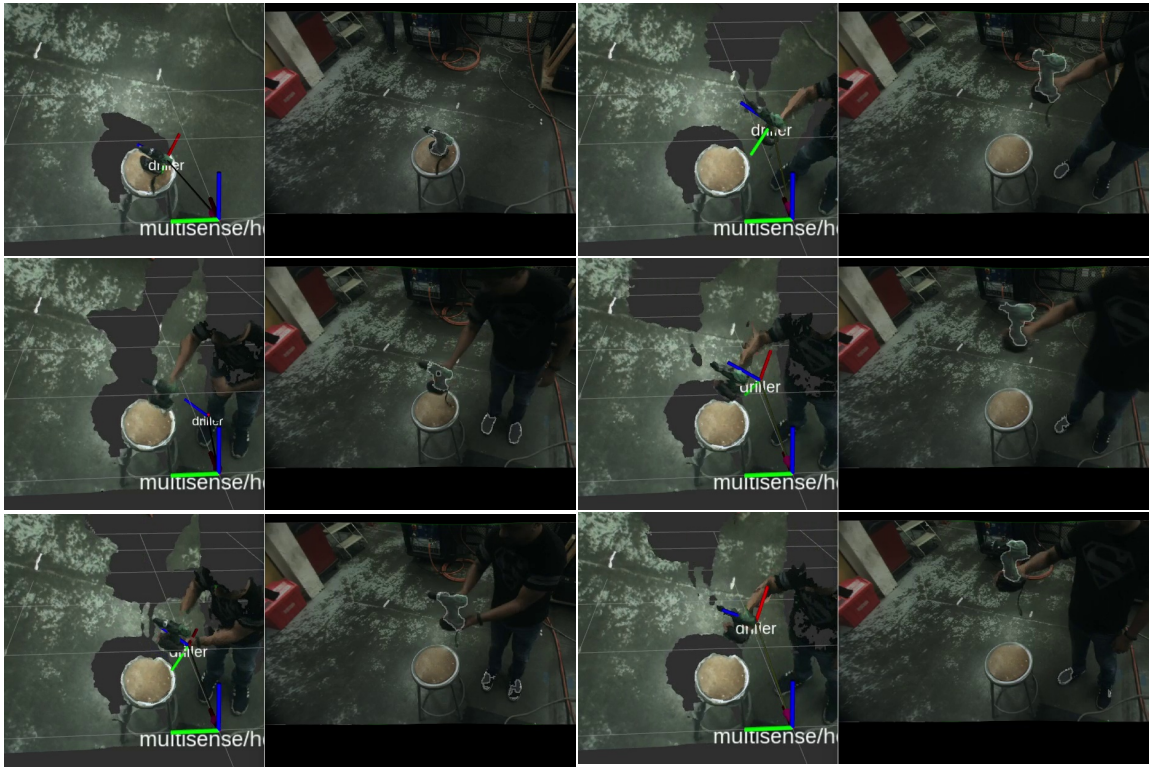


Figure 4.16: Sequence of images from a video of pose estimation experiment

Although the pipeline works very fast on the dataset as a standalone application, The integration with ROS and the use of an older GPU due to the limitations of the hardware connected with the Robot limits the performance. A Total average inference speed of 250 msec was obtained which is approximately 4Hz. This is a decent speed for any manipulation and planning frameworks to plan for trajectories.

4.2.6 Drill Pick-up Task

The drill Pick-up task is an extension of the drill detection task. This experiment is performed with a human in loop to verify every stage of the task being done correctly and approve the decisions made by the robot during the experiment.

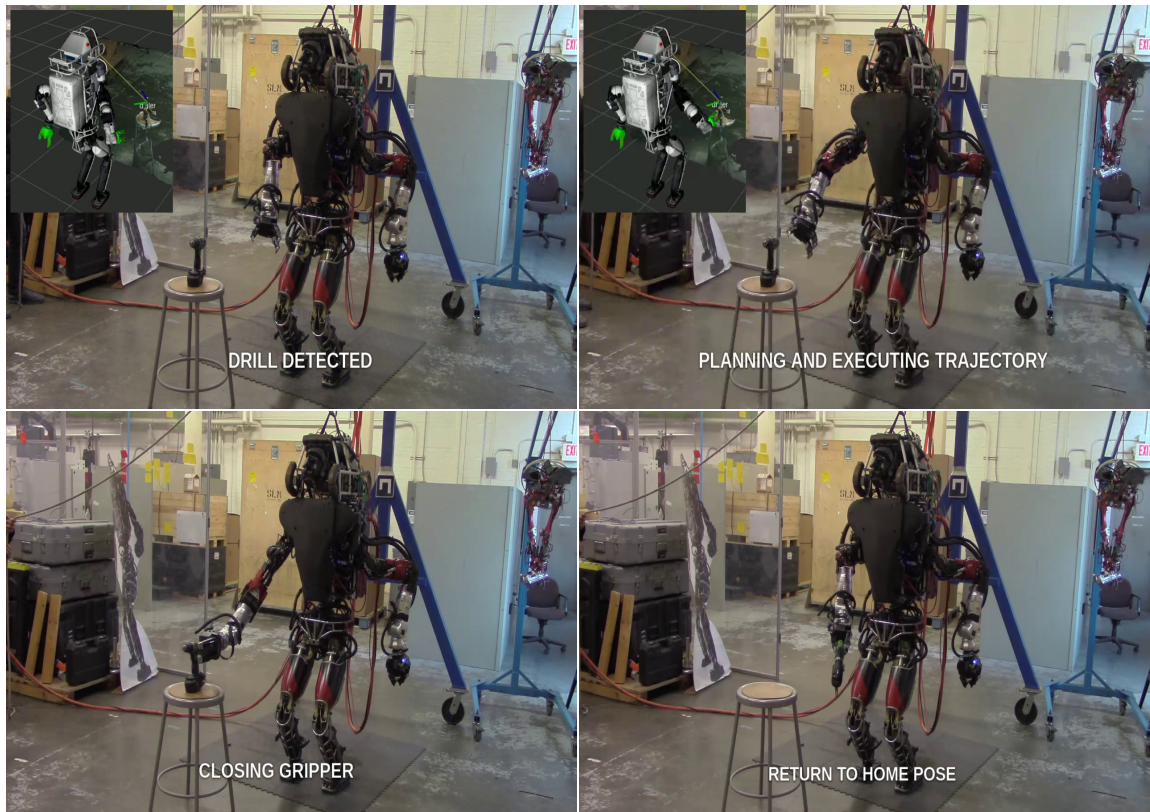


Figure 4.17: From Top left to bottom right: object detected by the robot, planning and executing trajectory to grab the object, closing the gripper to grasp, returning to default pose

In this experiment, the robot was initialized and calibrated, and put into STAND mode as the purpose of this experiment was only to manipulate the object and not to navigate around. The drill object was kept at a reachable distance with proper safety precautions taken during the high power and high pressure was activated. The experiment was conducted with a 7 DoF and a 10 DoF planner. For the 7Dof planner experiment, the object was placed close to the robot and could be easily picked. In pick-up task, the robot first detected the

object and its pose. It then planned a trajectory and waited for the user to approve for execution. Once the action was approved it moved the arm to the given pose and grasped the object. The operator then approved the grasps and approved return to home pose. refer 4.17

Out of 5 trials 4 were successful and 1 failed due to the error in the achieved task space position of the end-effector. This can be caused due to an error in the plan or errors in the intrinsic sensor such as joint encoders.

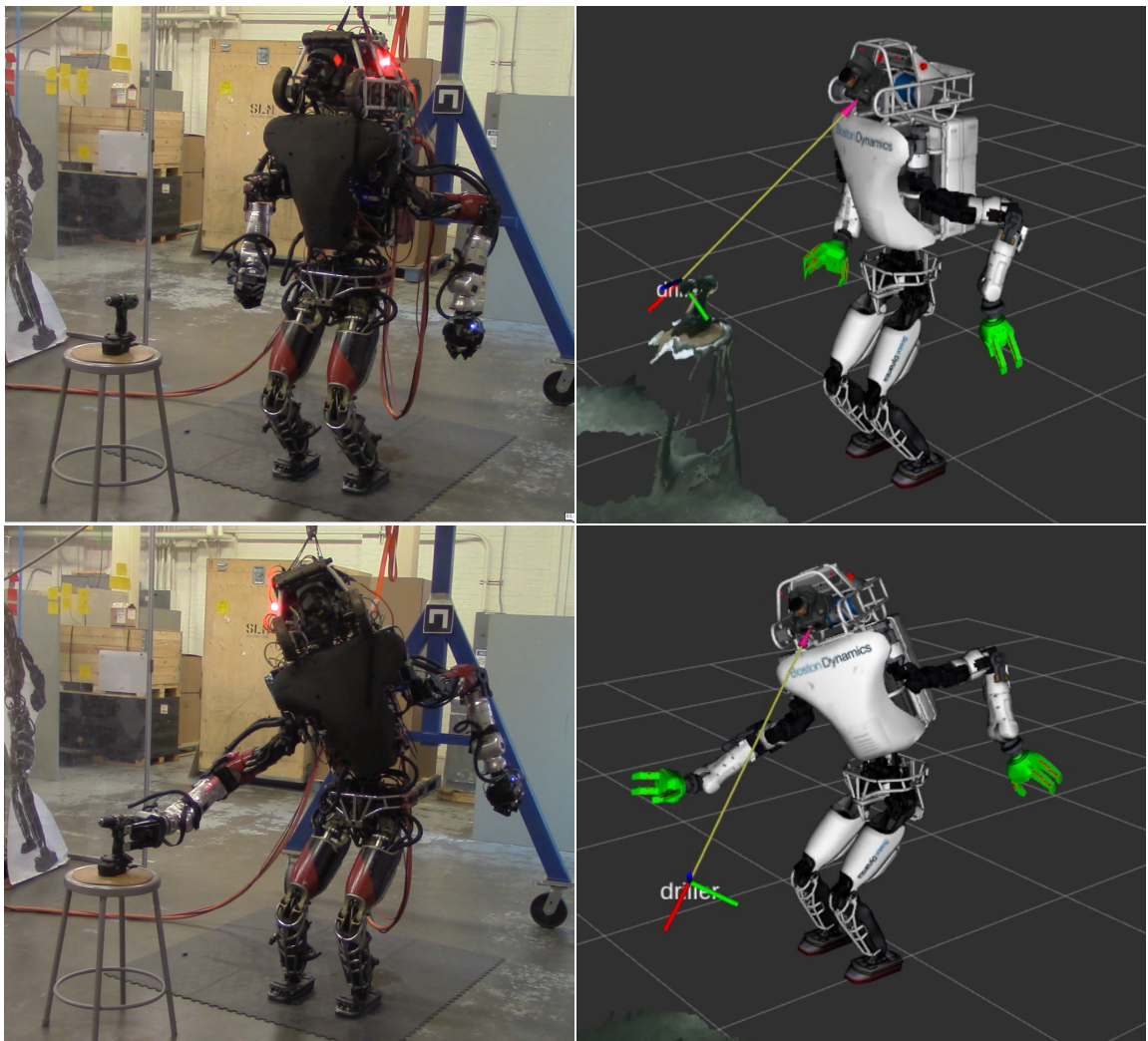


Figure 4.18: Failing to continuously detect object with 10DoF planning experiment

Similarly, a 10-DoF planner was used to pick-up the object in the scene. The object

was placed at a farther distance than the previous experiment such that the robot plans a trajectory using all the 10 torso joints. As the object was stationary the robot was able to detect and pick up the object but the experiment failed in continuously detecting the object. As the robot reaches to grab the object, it's head turns away from the object and out of the field of view for the object. Thus the network starts to give random values for 6DoF Pose which can be seen in 4.18. The robot still achieves a perfect grasp but would fail if the object moved during the execution of the task.

Chapter 5

Conclusion

5.1 Conclusion

In this work, two feature-based approaches were evaluated on the basis of their inference speed and accuracy. The approach with point cloud segmentation and using feature histograms to detect objects was observed to be affected by the symmetry of objects and outliers. Moreover, the segmentation of the point cloud took significant time to process which makes it inefficient for real-time applications. On the contrary, the projection based method was significantly faster and was able to extract object point clouds from the scene with very less false positives.

Based on the Conclusions derived from chapter 2, A 2 stage Convolutional Neural Network architecture was proposed which leverages the projection based approach. A pose estimation network was developed and evaluated on LINEMOD dataset to benchmark the performance of the pipeline.

To test the transferability of the framework, the proposed pipeline was used on a real ATLAS robot for object pick-up task. As the distribution of images in LINEMOD and the images captured by the MultiSense were different, a small dataset was created to fine-tune the object localization network which uses semantic segmentation proposals. The segmentation network shows no signs of over-fitting and gives an accuracy (IoU) of 0.9378 on the real world images. The pipeline is observed to work well for 4 out of 5 experiments

of detecting and picking up an object. It was observed that the pose network gives random values when the object is not present in the field of view. Thus, it cannot be directly used in closed loop manipulation frameworks.

Chapter 6

Future Work

- **Pose Estimation Network:** The projection based network needs a lot of pre-processing and is dependent on the spatial organization of information i.e organized point cloud. Some information is lost when the depth image is downsampled. This can be overcome by using a point cloud processing architectures such as PointNet[30] to directly regress the object pose.
- **Training and validation on YCB dataset** YCB[11] dataset contains multiple household objects with its ground truth pose and segmentation masks. As the dataset contains multiple masks and their instances, the SegNet can be replaced with an instance segmentation network such as MASK-RCNN [14]
- **Data generation with pose labels:** The dataset generated by combining and augmenting images from multisense sensor does not contain pose information. A more elaborate dataset can be generated by using a 3D model of the object and randomly translating in the scene. The projection of the object after translation can then be used to generate masks automatically. This will enable an end-to-end training of new architectures. Capturing the model of the object in itself can be a turntable experiment.

Appendix A

Appendix

A.1 TOUGH API description

Source code for simplified version of manipulation node used for object pick-up experiment using ATLAS Robot.

```

1 #include <pluginlib/class_loader.h>
2 #include <ros/ros.h>
3 #include <tf/tf.h>
4 #include <geometry_msgs/PoseStamped.h>
5 #include "tough_common/tough_common_names.h"
6 #include "tough_moveit_planners/taskspace_planner.h"
7 #include "tough_controller_interface/wholebody_control_interface.h"
8 #include <tough_controller_interface/gripper_control_interface.h>
9 #include <stdlib.h>
10 #include <std_msgs/String.h>
11 #include <tough_common/tough_common_names.h>
12
13 WholebodyControlInterface* wb_controller;
14 ChestControlInterface* chest_controller;
15 ArmControlInterface* arm_controller;
16 TaskspacePlanner* planner;
17 RobotDescription* rd_;
18 GripperControlInterface* gc_;
19 geometry_msgs::PoseStamped goal;

```

```
20
21 bool RUN_STATUS = true;
22
23 enum class PLANNER{
24     7DOF,
25     10DOF
26 };
27
28 bool moveToPose(const std::string planning_group, geometry_msgs::PoseStamped&
    goal)
29 {
30     moveit_msgs::RobotTrajectory trajectory_msg;
31     if (planner->getTrajectory(goal, planning_group, trajectory_msg))
32     {
33         wb_controller->executeTrajectory(trajectory_msg);
34         return true;
35     }
36     return false;
37 }
38
39 void poseCB(geometry_msgs::PoseStamped &msg)
40 {
41     goal = msg;
42 }
43
44 void waitForUser()
45 {
46     int choice;
47     std::cout << "Press 0 to abort, Press any other key to continue.." << std
        ::endl;
48     std::cin >> choice;
49     if (choice==0)
50         RUN_STATUS = false;
51 }
52
53 int main(int argc, char** argv)
54 {
```

```

55  ros::init(argc, argv, "motion_planning_tutorial");
56  ros::AsyncSpinner spinner(1);
57  spinner.start();
58  ros::NodeHandle nh;
59
60  ros::Subscriber sub = nh.subscribe("drill_detector", 1000, poseCB);
61
62  wb_controller = new WholebodyControlInterface(nh);
63  planner = new TaskspacePlanner(nh);
64  chest_controller = new ChestControlInterface(nh);
65  arm_controller = new ArmControlInterface(nh);
66  rd_ = RobotDescription::getRobotDescription(nh);
67  gc_ = new GripperControlInterface(nh);
68
69  RobotSide inputSide = RobotSide::RIGHT; // Right hand used
70  PLANNER planner_ = PLANNER::7DOF; // 7DoF planner used
71  std::string planning_group;
72  gc_>openGripper(inputSide);
73  waitForUser();
74
75  while (RUN_STATUS)
76  {
77      if (inputSide == RobotSide::LEFT)
78      {
79          if (planner_==PLANNER::7DOF)
80          {
81              planning_group.assign(TOUGH_COMMON_NAMES::LEFT_ARM_7DOF_GROUP);
82          }
83          else
84          {
85              planning_group.assign(TOUGH_COMMON_NAMES::LEFT_ARM_10DOF_GROUP);
86          }
87      }
88      else
89      {
90          if (planner_==PLANNER::7DOF)
91          {

```



```
92     planning_group.assign(TOUGH_COMMON_NAMES::RIGHT_ARM_7DOF_GROUP);
93 }
94 else
95 {
96     planning_group.assign(TOUGH_COMMON_NAMES::RIGHT_ARM_10DOF_GROUP);
97 }
98 }
99 gc_>openGripper(inputSide);
100 waitForUser();
101 moveToPose(planning_group, goal);
102 waitForUser();
103 gc_>closeGripper(inputSide);
104 waitForUser();
105 arm_controller->moveToDefaultPose(side, 3.0);
106 chest_controller->resetPose();
107 }
108 // clean-up
109 delete wb_controller;
110 delete chest_controller;
111 delete arm_controller;
112 delete planner;
113 delete rd_;
114 delete gc_;
115 }
```

Listing A.1: TOUGH EXAMPLE

Bibliography

- [1] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, and Senior Member. SegNet : A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *Arxiv*, pages 1–14, 2017.
- [2] Alexander Buslaev, Alex Parinov, Eugene Khvedchenya, Vladimir I. Iglovikov, and Alexandr A. Kalinin. Albumentations: fast and flexible image augmentations. *Arxiv*, 2018.
- [3] Zhe Cao, Yaser Sheikh, and Natasha Kholgade Banerjee. Real-time scalable 6DOF pose estimation for textureless objects. *Proceedings - IEEE International Conference on Robotics and Automation*, 2016-June:2441–2448, 2016.
- [4] Liang Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.
- [5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [6] Thanh-Toan Do, Ming Cai, Trung Pham, and Ian Reid. Deep-6DPose: Recovering 6D Object Pose from a Single RGB Image. *Arxiv*, 2018.
- [7] Elan Dubrofsky. Homography Estimation. *Opt. Eng.*, 15(March):977, 2009.
- [8] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and

- Wolfram Burgard. Multimodal deep learning for robust RGB-D object recognition. *IEEE Int. Conf. Intell. Robot. Syst.*, 2015-Decem:681–687, 2015.
- [9] Jagtap V. et. al. Tough. *GitHub repository*, 2019.
- [10] William Woodall. et. al. rviz. *GitHub repository*, 2019.
- [11] et. al. Berk Calli. YCB Object Set. *online*, 2015.
- [12] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A Review on Deep Learning Techniques Applied to Semantic Segmentation. *Arxiv*, pages 1–23, 2017.
- [13] Philipp Glira. Iterative closest point (icp), 2016. Last accessed 11 April 2019.
- [14] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-October:2980–2988, 2017.
- [15] Marti A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, July 1998.
- [16] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(5):876–888, May 2012.
- [17] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 1 Gradient Response Maps for Real-Time Detection of Texture-Less Objects. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, pages 1–14, 2012.
- [18] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In Kyoung Mu Lee, Yasuyuki

- Matsushita, James M. Rehg, and Zhanyi Hu, editors, *Computer Vision – ACCV 2012*, pages 548–562, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [19] Omid Hosseini Jafari, Siva Karthik Mustikovela, Karl Pertsch, Eric Brachmann, and Carsten Rother. iPose: Instance-Aware 6D Pose Estimation of Partly Occluded Objects. *Arxiv*, 2017.
- [20] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:1530–1538, 2017.
- [21] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A Large-Scale Hierarchical Multi-View RGB-D Object Dataset. *Arxiv*, 2011.
- [22] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. EPnP: An accurate $O(n)$ solution to the PnP problem. *Int. J. Comput. Vis.*, 81(2):155–166, 2009.
- [23] David G. Lowe. Object Recognition from Local Scale-Invariant Features. *Arxiv*, pages 35–40, 1999.
- [24] Frank Michel, Alexander Kirillov, Eric Brachmann, Alexander Krull, Stefan Gumhold, Bogdan Savchynskyy, and Carsten Rother. Global Hypothesis Generation for 6D Object Pose Estimation. *CVPR*, pages 462–471, 2016.
- [25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *online*, 2017.
- [26] Neil D. McKay Paul J. Besl. A Method for Registration of 3-D Shapes. *online*, 1992.
- [27] Georgios Pavlakos, Xiaowei Zhou, Aaron Chan, Konstantinos G. Derpanis, and Kostas Daniilidis. 6-DoF object pose from semantic keypoints. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2011–2018, 2017.

- [28] Thomas Petersen. A Comparison of 2D-3D Pose Estimation Methods. *Master's thesis, Aalborg Univ. Media Technol. Comput. Vis. Graph. Laurrupvang*, 15:2750, 2008.
- [29] Edgar Riba Pi, Francesc Moreno, Noguera Adrián, and Peñate Sanchez. Universitat Politècnica de Catalunya Implementation of a 3D pose estimation algorithm. *online*, 0(June), 2015.
- [30] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *CVPR2017*, pages 601–610, 2017.
- [31] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.
- [32] Mahdi Rad and Vincent Lepetit. BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-October:3848–3856, 2017.
- [33] Carnegie Robotics. MultisenseSL. <https://carnegierobotics.com/>, 2012.
- [34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9351:234–241, 2015.
- [35] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3D recognition and pose using the viewpoint feature histogram. *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pages 2155–2162, 2010.
- [36] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. pages 1–14, 2014.
- [37] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-Time Seamless Single Shot 6D Object Pose Prediction. *Arxiv*, 2017.

- [38] Sik-Ho Tsang. Review: Deepmask (instance segmentation), 2018. Last accessed 11 April 2019.
- [39] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion. *Arxiv*, 2019.
- [40] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *Arxiv*, 2017.
- [41] Nida M Zaitoun and Musbah J Aql. Survey on Image Segmentation Techniques. *Procedia - Procedia Computer Science*, 65(Iccmit):797–806, 2015.
- [42] Andy Zeng, Kuan Ting Yu, Shuran Song, Daniel Suo, Ed Walker, Alberto Rodriguez, and Jianxiong Xiao. Multi-view self-supervised deep learning for 6D pose estimation in the Amazon Picking Challenge. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1386–1393, 2017.