SCALABLE, PHYSICS-AWARE 6D POSE ESTIMATION FOR ROBOT MANIPULATION

By

CHAITANYA MITASH

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Abdeslam Boularias, Kostas E. Bekris

And approved by

Abdeslam Boularias
Kostas E. Bekris
Ahmed Elgammal
Henrik I. Christensen

New Brunswick, New Jersey

October 2020
Robot Manipulation often depend on some form of pose estimation to represent the state of the world and allow decision making both at the task-level and for motion or grasp planning. Recent progress in deep learning gives hope for a pose estimation solution that could generalize over textured and texture-less objects, objects with or without distinctive shape properties, and under different lighting conditions and clutter scenarios. Nevertheless, it gives rise to a new set of challenges such as the painful task of acquiring large-scale labeled training datasets and of dealing with their stochastic output over unforeseen scenarios that are not captured by the training. This restricts the scalability of such pose estimation solutions in robot manipulation tasks that often deal with a variety of objects and changing environments.

The thesis first describes an automatic data generation and learning framework to address the scalability challenge. Learning is bootstrapped by generating labeled data via physics simulation and rendering. Then it self-improves over time by acquiring and labeling real-world images via a search-based pose estimation process. The thesis proposes algorithms to generate and validate object poses online based on the objects’ geometry and based on the physical consistency of their scene-level interactions. These algorithms provide robustness even when there exists a domain gap between the synthetic training and the real test scenarios. Finally, the thesis proposes a manipulation planning framework that goes beyond model-based pose
estimation. By utilizing a dynamic object representation, this integrated perception and manip-
ulation framework can efficiently solve the task of picking unknown objects and placing them
in a constrained space.

The algorithms are evaluated over real-world robot manipulation experiments and over
large-scale public datasets. The results indicate the usefulness of physical constraints in both
the training and the online estimation phase. Moreover, the proposed framework, while only
utilizing simulated data can obtain robust estimation in challenging scenarios such as densely-
packed bins and clutter where other approaches suffer as a result of large occlusion and ambi-
guities due to similar looking texture-less surfaces.
Acknowledgements

I would like to thank my advisers Abdeslam Boularias and Kostas Bekris for inspiring and guiding me in my research. I am also thankful to my thesis committee members Prof. Ahmed Elgammal and Prof. Henrik Christensen for their insightful comments.

I have been fortunate to have collaborated (formally or informally) with several of my lab members. I would like to thank them all. They have been a great company and have always helped me during need.

I would like to thank my funding agencies (NSF and JD.com) and all my colleagues during the summer internships at Microsoft Hololens and Amazon Robotics. Their support and interactions were critical to my research.

This journey would not have been possible without the love, support and trust from my friends and family.
# Table of Contents

Abstract ................................................................. ii
Acknowledgements ..................................................... iv
List of Tables ........................................................... xi
List of Figures .......................................................... xiii

1. Introduction .......................................................... 1
   1.1. Self-supervised Learning for Object Detection using Physics-based Simulation 2
   1.2. Robust Pose Estimation with Stochastic Congruent Sets ................................. 3
   1.3. Joint Pose Estimation via Physics-aware Monte Carlo Tree Search ..................... 4
   1.4. Scene-level Pose Estimation for Multiple Instances of Densely Packed Objects ........... 5
   1.5. Applications of Model-based 6D Pose Estimation ........................................... 6
   1.6. Task-driven Perception and Manipulation for Constrained Placement of Unknown Objects ........................................ 7

2. Related Work .......................................................... 9
   2.1. Pose estimation based on Manually-defined Feature Descriptors ......................... 9
   2.2. Learning-based Pose Estimation ......................................................................... 10
   2.3. Point-set Registration Methods .......................................................................... 11
   2.4. Global Scene-Level Reasoning ......................................................................... 11
   2.5. Object Representation for Robot Manipulation ................................................. 12

   3.1. Physics-aware Synthetic Data Generation ......................................................... 16
   3.2. Self-Learning via Multi-view Pose Estimation ................................................... 19
3.3. Evaluation

3.3.1. Datasets

3.3.2. Evaluating the Object Detector trained in Simulation

Generating training data using test data distribution
Uniformly sampled synthetic data
Generating training data with physics-aware simulation

3.3.3. Evaluating Self-learning

3.3.4. Evaluating the detector for 6DoF Pose estimation

4. Robust Pose Estimation with Stochastic Congruent Sets

4.1. Problem statement

4.2. Defining the Segmentation-based Prior

4.3. Congruent Set Approach for Computing the Best Transform

4.4. Stochastic Optimization for Selecting the Base

4.5. Evaluation

4.5.1. Amazon Picking Challenge (APC) dataset

4.5.2. Computational cost

4.5.3. YCB-Video dataset

5. Joint Pose Estimation in Clutter via Physics-aware Monte Carlo Tree Search

5.1. Problem Setup

5.2. Hypothesis Generation

5.3. Clustering of Hypotheses

5.4. Search

5.5. Evaluation

5.5.1. Dataset and Evaluation Metric

5.5.2. Pose Estimation without Search

5.5.3. Pose Estimation with the proposed approach

5.5.4. Evaluating over Benchmark for Pose Estimation

5.5.5. Limitations
6. Scene-level Pose Estimation for Multiple Instances of Densely Packed Objects .......................... 61
   6.1. Semantic and Boundary Predictions ................................................. 63
   6.2. Geometry-aware pose sampling ..................................................... 65
   6.3. Pose Hypothesis Quality Evaluation ................................................ 68
   6.4. Scene-level Pose Selection ........................................................... 70
   6.5. Experiments ................................................................. 72
      6.5.1. Implementation details ......................................................... 73
      6.5.2. Experiments on bin-picking dataset .......................................... 74
      6.5.3. Experiments on densely-packed dataset ...................................... 74
      6.5.4. Experiments on occluded-linemod dataset .................................. 78
      6.5.5. Evaluating semantic and visibility boundary prediction ................... 79
      6.5.6. Learned vs. Manually-defined objective functions ......................... 80
      6.5.7. Ablation study ................................................................. 82
      6.5.8. Computation time ............................................................... 83

7. Applications of Model-based 6D Pose Estimation .................................................. 84
   7.1. Robust Product Packing ............................................................... 84
      7.1.1. Problem Setup and Notation ................................................... 86
      7.1.2. System Components ............................................................. 88
      7.1.3. Proposed Solution .............................................................. 89
      7.1.4. Baseline: Pose Estimation and Picking ....................................... 90
      7.1.5. Toppling ................................................................. 90
      7.1.6. Point Cloud Driven Adaptive Pushing ....................................... 92
      7.1.7. Fine Correction using Push and Pull Primitives ............................ 92
      7.1.8. Evaluation ................................................................. 93
   7.2. In-hand Pose estimation ............................................................ 96
      7.2.1. Problem Formulation ............................................................ 96
      7.2.2. Hand State Estimation .......................................................... 97
      7.2.3. Object Pose Hypotheses Generation and Clustering ....................... 99
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4.10.</td>
<td>YCB-Video Dataset</td>
<td>128</td>
</tr>
<tr>
<td>7.4.11.</td>
<td>Occluded Linemod Dataset</td>
<td>128</td>
</tr>
<tr>
<td>7.5.</td>
<td>Safely Picking Objects in Clutter</td>
<td>129</td>
</tr>
<tr>
<td>7.5.1.</td>
<td>Generation of Discrete Pose Distributions</td>
<td>130</td>
</tr>
<tr>
<td>7.5.2.</td>
<td>Problem setup and notation</td>
<td>132</td>
</tr>
<tr>
<td>7.5.3.</td>
<td>Algorithmic Framework</td>
<td>134</td>
</tr>
<tr>
<td>7.5.4.</td>
<td>Challenge - Dynamic programming does not hold</td>
<td>136</td>
</tr>
<tr>
<td>7.5.5.</td>
<td>MaxSuccess Search</td>
<td>136</td>
</tr>
<tr>
<td>7.5.6.</td>
<td>Experimental Setup</td>
<td>139</td>
</tr>
<tr>
<td>7.5.7.</td>
<td>Simulation Experiments</td>
<td>139</td>
</tr>
<tr>
<td>7.5.8.</td>
<td>Real-world Experiments</td>
<td>141</td>
</tr>
<tr>
<td>8.</td>
<td>Task-driven Perception and Manipulation for Constrained Placement of Unknown Objects</td>
<td>144</td>
</tr>
<tr>
<td>8.1.</td>
<td>Problem Setup and Notation</td>
<td>147</td>
</tr>
<tr>
<td>8.2.</td>
<td>Proposed pipeline</td>
<td>149</td>
</tr>
<tr>
<td>8.3.</td>
<td>System Design &amp; Implementation</td>
<td>150</td>
</tr>
<tr>
<td>8.4.</td>
<td>Experimental Setup</td>
<td>154</td>
</tr>
<tr>
<td>8.5.</td>
<td>Results</td>
<td>155</td>
</tr>
<tr>
<td>8.6.</td>
<td>Judging the Intent of Pointing Actions with Robotic Arms</td>
<td>159</td>
</tr>
<tr>
<td>8.6.1.</td>
<td>Communicating Pick-and-Place</td>
<td>159</td>
</tr>
<tr>
<td>8.6.2.</td>
<td>Experiment Setup</td>
<td>162</td>
</tr>
<tr>
<td>8.6.3.</td>
<td>Data Collection</td>
<td>164</td>
</tr>
<tr>
<td>8.6.4.</td>
<td>Experimental Conditions</td>
<td>164</td>
</tr>
<tr>
<td>8.6.5.</td>
<td>Analysis</td>
<td>168</td>
</tr>
<tr>
<td>8.6.6.</td>
<td>Human Evaluation of Instructions</td>
<td>171</td>
</tr>
<tr>
<td>8.6.7.</td>
<td>Design Principles</td>
<td>171</td>
</tr>
<tr>
<td>9.</td>
<td>Discussion</td>
<td>174</td>
</tr>
<tr>
<td>9.1.</td>
<td>Future Work</td>
<td>177</td>
</tr>
</tbody>
</table>
# List of Tables

3.1. Evaluation on Princeton’s Shelf&Tote dataset [1] ........................................ 24  
3.2. Detection accuracy on table-top experiments ................................................. 25  
3.3. Comparing the performance of the proposed system to state-of-the-art tech- 
niques for pose estimation. ...................................................... 26  
4.1. Average rotation error, translation error and execution time (per object) over 
APC dataset ................................................................. 35  
4.2. Computation complexity for the different components of the registration process. 35  
4.3. Success given the area under the accuracy-threshold curve and computation 
time (per object) on the YCB-Video dataset. ........................................ 37  
5.1. Evaluating the quality of the hypotheses set returned by Super4CPS [2] with 
respect to different metrics. ...................................................... 45  
5.2. Comparing the accuracy of MCTS with different pose estimation techniques . 53  
5.3. Evaluating pose recall rate on the LINEMOD dataset according to the recent 
benchmark[3] ................................................................. 58  
5.4. Evaluating pose recall rate on the LINEMOD-Occluded dataset according to 
the recent benchmark[3] ...................................................... 59  
6.1. Description of features indicating good pose alignment with sensory input ........ 69  
6.2. Pose estimation recall on the bin-picking dataset. ....................................... 74  
6.3. Pose retrieval recall rate on densely-packed dataset. ................................... 75  
6.4. Pose recall on the occluded-linemod dataset. .......................................... 76  
6.5. Alternative strategies considered during development. .............................. 77  
6.6. Evaluating boundary prediction. ...................................................... 77  
6.7. Evaluating different objective functions for pose selection. ....................... 79  
6.8. Effect of using boundaries for hypotheses generation. .............................. 82
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.9</td>
<td>Evaluating different training strategies.</td>
</tr>
<tr>
<td>6.10</td>
<td>Computation time for different components of the pipeline.</td>
</tr>
<tr>
<td>7.1</td>
<td>Pose recall percentage on proposed dataset</td>
</tr>
<tr>
<td>7.2</td>
<td>Ablation study of critical components in our system</td>
</tr>
<tr>
<td>7.3</td>
<td>Comparing the performance of se(3)-TrackNet (Gray) with state-of-the-art techniques on YCB Video</td>
</tr>
<tr>
<td>7.4</td>
<td>Results evaluated on YCBInEOAT-dataset</td>
</tr>
<tr>
<td>7.5</td>
<td>Area under the accuracy-threshold curve for 6D Pose estimation on YCB-Video dataset (ADD-S metric) and ADD metric for Occluded Linemod with threshold of 10% of the diameter (ADD-S metric for 2 objects).</td>
</tr>
<tr>
<td>7.6</td>
<td>Path cost and planning time.</td>
</tr>
<tr>
<td>8.1</td>
<td>Evaluating the task success rate of the proposed manipulation pipeline</td>
</tr>
<tr>
<td>8.2</td>
<td>Comparing the quality and computation time for the solutions found with the baseline and the proposed approach</td>
</tr>
<tr>
<td>8.3</td>
<td>Results of the unnatural scene and natural scene. (Numbers are out of 30.)</td>
</tr>
</tbody>
</table>
## List of Figures

1.1. Research overview .................................................. 2
3.1. Introduction: Learning object detector via physics simulation ........ 15
3.2. Pipeline for physics aware simulation .......................... 16
3.3. Automatic self-labeling pipeline ................................. 19
3.4. Manipulator performing scene reconfiguration for data collection .... 21
3.5. Examples of training data generated from different techniques ........ 22
3.6. Ablation study for training the object detector .................. 25
4.1. A motivating example for pointset registration based on soft segmentation . 28
4.2. Stochastic optimization for selecting the base set ................. 31
4.3. Ablation study for registration techniques ........................ 36
5.1. Setup for scene-level reasoning ................................... 40
5.2. Pose hypothesis generation and clustering ........................ 43
5.3. Congruent set matching process .................................. 45
5.4. Monte Carlo Tree Search expansion process ...................... 52
5.5. (Left) Rotation error in degrees and (Right) Translation error in cm as a function of the number of iterations. .......................... 55
5.6. Qualitative results on Extended Rutgers RGBD dataset ............. 56
5.7. Qualitative results on Linemod and Linemod-Occluded dataset ....... 57
6.1. System pipeline and example output of the proposed approach on densely-packed scenes ................................................ 62
6.2. CNN architecture for semantic classes and boundary prediction ........ 63
6.3. Sampling a base in multi-instance scenarios ...................... 66
6.4. Experiment scenarios ............................................... 72
6.5. Evaluating semantic segmentation ................................ 78
6.6. Examples of synthetic training data and boundary predictions on real images. . 78
6.7. Learned feature importance .................................................. 81
6.8. Pose recall as a function of the threshold for the learned objective function. . 81
6.9. Recall as a function of the number of pose candidates and the dispersion parameter $\gamma$. . 82
7.1. The product packing problem for cuboid products .......................... 86
7.2. Experiment setup ................................................................. 86
7.3. Pipeline in terms of control, data flow and failure handling .................. 87
7.4. Adaptive pushing .............................................................. 91
7.5. Fine layout adjustment and correction. ..................................... 92
7.6. Qualitative results from packing pipelines .................................... 94
7.7. In-hand pose estimation ....................................................... 96
7.8. Pipeline for Hand-Object Pose estimation ................................. 97
7.9. Adaptive hand with 2 underactuated fingers. .............................. 97
7.10. Sampling pose transformations .............................................. 100
7.11. Comparison on simulation dataset ........................................ 102
7.12. Objects used in experiments ................................................ 103
7.13. Qualitative results for the proposed approach ............................ 105
7.15. Performance of the proposed pose tracking framework ................... 108
7.16. Object pose tracking overview ............................................. 109
7.17. Network architecture for $se(3)$-TrackNet. ............................... 110
7.18. Comparison of Domain Randomization against PPDR (Physically Plausible
      Domain Randomization) ..................................................... 112
7.19. Dataset collection setup ..................................................... 116
7.20. Qualitative results ......................................................... 117
7.21. Ablation study on critical components of proposed framework. ........... 119
7.22. Pose estimation with network trained over weakly annotated images ....... 120
7.23. Network architecture ....................................................... 121
7.24. Performance analysis for object detection ............................... 126
7.25. Pose estimation recall on Occluded-Linemod dataset ........................................... 129
7.26. Discrete distribution of object poses ................................................................. 130
7.27. Perception pipeline ............................................................................................. 131
7.28. Computing the prospect of a path accurately reaching the target ......................... 135
7.29. Example showing locally optimal paths does not turn out to be globally optimal. 136
7.30. Number of collisions at different noise levels ...................................................... 138
7.31. Success rate for different algorithms in all 4 benchmarks. .................................. 140
7.32. Experimental results on real vision system in different scenarios ......................... 141
7.33. Real system setup ............................................................................................... 142
7.34. Qualitative result ............................................................................................... 143
8.1. Pick-handoff-place demonstration over unknown object ........................................ 145
8.2. Dynamic object representation gets updated during manipulation ......................... 146
8.3. Proposed manipulation pipeline ........................................................................... 148
8.4. Hardware Setup. ................................................................................................ 151
8.5. Experiment setup ................................................................................................ 153
8.6. Qualitative results indicating different solution modes of the proposed pipeline. 156
8.7. Split of outcomes of experiments within success and various failure cases for each category. ........................................................................................................... 157
8.8. Demonstrations of the proposed pipeline’s operation (left) in the presence of shape ambiguity (right) on the object flipping task. ............................................. 159
8.9. Referential vs locating pointing ............................................................................ 160
8.10. Workspace setup ............................................................................................... 161
8.11. Illustrations of pointing actions in simulation ...................................................... 163
8.12. Natural vs. Unnatural scenes .............................................................................. 164
8.13. A cluttered trial .................................................................................................... 166
8.14. Object placement configurations ......................................................................... 167
8.15. The aggregated results from the referential versus spatial trials for the Baxter and Kuka robots ........................................................................................................ 168
8.16. Ambiguity in clutter ............................................................................................ 170
9.1. Application scenarios ... .............................. 176
Chapter 1
Introduction

Object recognition and pose estimation are fundamental problems in computer vision and have significant applications in the areas of robot manipulation and augmented reality. Typically, the task involves localizing one or many objects of interest in an image and estimating their 3D position and 3D orientation in a pre-determined reference frame, such as the camera or a robot.

There has been significant progress in object recognition, given recent advances in deep learning technology. Nevertheless, these tools often require a large number of labeled training examples. This limits their applicability to fields like robotics, where solutions must scale to a large number of objects and a variety of conditions. Additionally, in many cases estimating the object pose becomes very challenging due to severe occlusions and the presence of several objects that are similar in appearance or geometry.

The focus of my thesis is on addressing the challenges in 6D object pose estimation in the context of robot manipulation. It does so by leveraging geometric priors and physical constraints, in both the learning process and the online estimation. The proposed learning process is scalable as the training is performed exclusively over simulated data. The solution is physics-aware as it utilizes scene-level interactions between the objects and with the surrounding surfaces in the data generation process and during the joint pose estimation step. Additionally, to deal with objects for which no shape prior is available, this work provides a representation that allows manipulation planning and simultaneous reasoning about the object’s shape and pose in the context of the task.

Figure 1.1 illustrates the structure of the thesis where the chapters are based on contributed publications [5,6,7,8,9,10,11,12,13,14,15,16]. Below is a brief overview of the motivation, contribution and results corresponding to each of these chapters.
1.1. Self-supervised Learning for Object Detection using Physics-based Simulation

Learning-based pipelines for object detection and pose estimation need access to a large number of training examples, which in turn requires manual labeling. This is not scalable in robotics scenarios due to the vast number of objects and a variety of conditions that can arise.

This chapter proposes an automated data generation pipeline [5, 6] to train a convolutional neural network (CNN) for object detection in clutter. Given access to 3D object models, several aspects of the environment are physically simulated. The models are placed in physically realistic poses with respect to their workspace to generate a labeled synthetic dataset. To further improve object detection, the network self-trains over real images that are automatically
labeled using a robust multi-view pose estimation process.

Experiments were performed over challenging scenarios in the Shelf&Tote benchmark dataset [1]. Intersection-over-Union (IoU) success rate is measured to evaluate object detection. The success rate improves from 31% to 70% when utilizing the physically-realistic training data as opposed to when training is performed using similar datasets that are physically unrealistic. Furthermore, it is observed that the self-learning process improves the accuracy of the detector over time (from 70% to 82%). Qualitatively, the process is able to learn new viewpoints and exhibits robustness to occlusion.

1.2. Robust Pose Estimation with Stochastic Congruent Sets

Despite the progress due to Convolutional Neural Networks (CNNs), object detection and semantic segmentation can be noisy, especially when the CNN is only trained on synthetic data. This leads to failure in estimating an accurate object pose.

A novel stochastic optimization process is proposed [7] that treats the segmentation output of CNNs as a confidence probability. The algorithm, called Stochastic Congruent Sets (StoCS), samples point-sets on the point cloud according to the soft segmentation distribution and so as to agree with the objects known geometry. The point-sets are then matched to congruent sets on the 3D object model to generate and validate pose estimates.

StoCS is evaluated on a table-top dataset with objects from the Amazon Picking Challenge (APC) and on the YCB benchmark dataset. StoCS is observed to be robust, despite the fact that the CNN is trained only with synthetic data. On the YCB dataset, it outperforms a recent deep learning framework for 6D pose estimation and alternative point-set matching techniques both in terms of accuracy and computational efficiency. It achieves a success of 90.1% (based on the area under the accuracy-threshold curve metric) with an average run-time of 0.59s.
1.3. Joint Pose Estimation via Physics-aware Monte Carlo Tree Search

A typical pose estimation pipeline starts by generating a set of individual candidate object poses based on object segmentation, followed by global point cloud registration. Often, the most likely candidate returned from this process is not the most accurate, especially in cases of high occlusion. This motivates a global optimization process for improving these poses by taking into account scene-level physical interactions between objects. It also implies that the Cartesian product of candidate poses for interacting objects must be explored so as to identify the best scene-level hypothesis.

A Monte Carlo Tree Search (MCTS) based optimization process [5, 8] is proposed for scene-level reasoning with physics-based priors. The MCTS-based algorithm searches over the Cartesian product of individual object pose candidates to find the optimal scene hypothesis. The optimization is performed with respect to a score that captures the pixel-wise similarity between rendered hypothesized scenes and the observed depth image. The search performs constrained local optimization for each of these candidate poses with physics-driven corrections and Iterative closest point (ICP) refinement. This helps in pruning the large search space and achieving accurate pose estimates faster. MCTS handles in a principled way the trade-off between fine-tuning the most promising poses and exploring new ones by using the Upper Confidence Bound (UCB) technique.

The MCTS-based algorithm shows significant improvement in the quality of pose estimation as a result of scene-level reasoning, when compared to pose estimation performed individually for each object. A table-top dataset was designed with varying level of physical dependencies between objects. On this dataset, the mean rotation error for object poses computed by local reasoning was 10.5 degrees. It dropped to 4.6 degrees by applying the scene-level optimization. The combination of MCTS pose estimation with the previously discussed self-learning process for object detection achieves state-of-the-art performance on the public Occluded-Linemod dataset.
1.4. Scene-level Pose Estimation for Multiple Instances of Densely Packed Objects

The techniques discussed in previous chapters, similar to most state-of-the-art pose estimation techniques, are developed and evaluated for setups where each object appears once and for relatively sparsely placed objects on tabletops. Pose estimation for multiple instances of the same object type and where objects may be densely packed or in highly unstructured but dense piles has received less attention, despite its significance in application domains, such as logistics. This is partly due to the increased difficulty of such setups.

This chapter [9] introduces key machine learning operations that allow the realization of robust, joint 6D pose estimation of multiple instances of objects either densely packed or in unstructured piles from RGB-D data. The first objective is to learn semantic and instance-boundary detectors without manual labeling. Learning instance or visibility boundary is crucial in such scenarios, where the depth image and object geometry are not sufficient to resolve the ambiguity in segmentation and pose estimation. An adversarial training framework in conjunction with physics-based simulation is proposed to achieve detectors that behave similarly over synthetic and unlabeled real data. From this network, a stochastic segmentation output is considered, which is similar to the one in Chapter [4] but with additional reasoning about instance boundaries. Object pose candidates are sampled based on this output. The second objective is to automatically learn a single score for each pose candidate that represents its quality in terms of explaining the scene. This learning is performed via a gradient boosted tree. The proposed method uses features derived from surface and boundary alignment between the observed scene and the object model placed at hypothesized poses. Scene-level, multi-instance pose estimation is then achieved by an integer linear programming (ILP) process that selects hypotheses that maximize the sum of the learned individual scores, while respecting constraints, such as avoiding collisions.

The proposed adversarial training technique improves the final recall for object poses by approximately 10%. It also generalizes well to datasets with multiple object instances and multiple object classes as opposed to other popular image-to-image translation networks. The
ILP-based scene-level pose estimation achieves a high recall rate (of 82.1%) on the very challenging task of estimating all instances of all object categories in an image. It was evaluated on a dataset that contains scenes with up to 20 object instances with many cases of large occlusion.

1.5. Applications of Model-based 6D Pose Estimation

This chapter highlights applications of the 6D pose estimation techniques discussed earlier. These applications have been developed in collaboration with other PhD students and their technical contributions are covered by their thesis.

*Product packing pipeline [10]:* A complete robotic solution is designed to pick objects from unstructured piles and tightly pack them in containers. The solution combines RGB-D pose estimation (based on Chapter 4) with manipulation primitives like toppling, robust placement, and corrective packing to achieve high success on the task. *The corresponding publication was nominated for the Best Automation Paper award of ICRA 2019.*

*In-hand Pose Estimation and Tracking [12, 16]:* Estimating and tracking the pose of objects in-hand during manipulation is critical for the successful execution of planned tasks. Pose estimation for such objects could leverage the physical constraints (similar to Chapter 5, 6) from the estimation of hand-pose [12] to perform robust point-set matching (as in Chapter 4), even in cases of large occlusion. A tracking approach is proposed in [16]. It leverages physically simulated data (as in Chapter 3) to train a novel deep learning architecture that can predict relative poses between consecutive frames with high accuracy and speed.

*Learning with Weak Supervision [11]:* Supervised learning for pose estimation requires labeled training data. Given that manual annotation is not scalable and training with synthetic data introduces a domain gap between the training and the test set, this work introduces a technique for pose estimation based on weak labels. Just based on a training dataset with labels of which objects are present in images, it can learn to predict probability maps for the location of objects in images. This is utilized to perform pose estimation with the stochastic congruent set matching algorithm as presented in Chapter 4.

*Planning under Pose Uncertainty [12]:* The focus of this work is to generate safe and effective motion plans for picking objects from clutter given the uncertainty from the pose estimation.
process. A convolutional neural network is used to detect the existence of objects in the scene alongside predicting pixel-wise semantic labels. This is followed by a stochastic congruent set matching process (as in Chapter 4) to generate pose hypotheses. The pose hypotheses are clustered and a discrete probability distribution is derived for object poses and their existence in the scene, which is then utilized for path planning.

1.6. Task-driven Perception and Manipulation for Constrained Placement of Unknown Objects

The pose estimation problem considered up to this point depends on a known geometric and textured model of the object. This assumption can be restrictive in some scenarios as it requires the generation of such accurate models for all the objects in a scene. There has been recent progress in robot manipulation that has dealt with the case of no prior object models in the context of relatively simple tasks, such as bin-picking. Existing methods for more constrained problems, however, such as deliberate placement in a tight region, depend more critically on shape information to achieve safe execution.

This chapter introduces an algorithmic framework [14] for solving constrained placement tasks over previously unseen objects. It performs manipulation planning simultaneously over a conservative and an optimistic estimate of the objects volume. The conservative estimate ensures that the manipulation is safe while the optimistic estimate guides the sensor-based manipulation process when no solution can be found for the conservative estimate. To maintain these estimates and dynamically update them during manipulation, objects are represented by a simple volumetric representation, which stores sets of occupied and unseen voxels.

The effectiveness of the proposed approach is demonstrated by developing a robotic system that picks a previously unseen object from a table-top and places it in a constrained space. The system comprises of a dual-arm manipulator with heterogeneous end-effectors and leverages hand-offs as a re-grasping strategy. Real-world experiments show that straightforward pick-sense-and-place alternatives frequently fail to solve pick-and-constrained placement problems. The proposed pipeline, however, achieves more than 95% success rate and faster execution times as evaluated over multiple physical experiments.
**Pick-and-Place Task Communication [13]**: Specifying pick-and-place tasks could be challenging in scenarios where objects are not known in advance. One alternative is to use pointing to specify the target object to be manipulated and to identify the placement constraints. A collaborative study [13] that identifies the interpretive principles of referential pointing (to identify objects) and location pointing (to identify locations) for communicating pick-and-place tasks.
Chapter 2

Related Work

This chapter discusses the different existing methodologies for object pose estimation, their limitations and relation to the work presented in this thesis.

2.1. Pose estimation based on Manually-defined Feature Descriptors

One popular approach to pose estimation is to match feature points between textured 3D models and images [17, 18, 19]. This requires, however, textured objects and good lighting conditions, which has motivated instead the use of range data. Some range-based techniques compute correspondences between local point descriptors on the observed scene and on the object CAD models. Once correspondences are established, robust estimators like generalized Hough transform [20] or RANSAC [21] are used to compute the rigid transform that is consistent with the majority of correspondences. Several local descriptors are available [22], such as signature of histograms of orientations (SHOT) [23], fast point feature histogram FPFH [24] and Spin Images [25]. There has also been work on improving the efficiency of RANSAC and Hough transform [26, 27]. Feature-based approaches can be extended to multi-view object recognition [28] and pose estimation [29] so as to increase accuracy relative to single frame estimates. This family of methods depends on local surface information, which is sensitive to the resolution and quality of sensor and model data. The features are often parametrized by the area of influence, which is not trivial to decide. The smaller area could lead to less discriminative features between different surfaces on the object, while a larger area could result in sensitivity to occlusion and noise.

One proposed way to counter these limitations is to use oriented point pair features [30] so
as to create a global object model in the form of a map that stores the model points that exhibit each feature. This map can then be used to match the features in the scene and to get the object pose through a fast voting scheme. This idea was later extended to incorporate color [31], geometric edge information [32] and visibility context [33]. Recently, point pair features were used for segmenting the scene into several clusters, where each cluster generates a separate pose hypothesis [34]. The votes are weighted based on the probability of visibility of model points. Recent work [35] uses a sampling strategy for scene points by reasoning about the size of the object model. The approach modifies the voting scheme to accommodate sensor noise by also voting in the neighboring bins. Point pair features have been criticized in some occasions for performance loss in the presence of background clutter, sensor noise and also due to their quadratic computational complexity.

Another category of methods for pose estimation is based on template matching, such as LINEMOD [36] and variants like [37]. This method is based on viewpoint sampling around a 3D CAD model and building templates for each viewpoint based on color gradient and surface normals, which are later matched to compute object pose. GPU-based implementations help to speed-up computation [38]. Template matching approaches often fail under occlusion scenarios.

2.2. Learning-based Pose Estimation

Initial approaches to learning-based pose estimation [39, 40, 41] are trained to predict 3D object coordinates for points belonging to the corresponding object in the scene. A recent effort [42] performs geometric validation on these predictions using a conditional random field. Inspired by the applicability of CNNs for descriptor learning of RGB-D views [43], recent work [44] has demonstrated deep learning of descriptive features from local RGB-D patches used to create 6D pose hypotheses. Similarly, CNNs have been used to detect semantic keypoints to estimate the 6 DoF pose consistent with the keypoints [45]. [46, 47] infer coordinates of 3d bounding boxes on the image and use perspective-n-point (PnP) algorithm to compute the pose. More recent approaches learn to directly regress the translation and rotation parameters for objects from RGB [48] or RGB-D images [49]. Most approaches depend on object detection or semantic
segmentation \cite{50,51,52} as an initial step. The success of these learning-based approaches depend critically on access to representative labeled training data that is hard to acquire.

An alternative to collecting and labeling training data is to use data generated in simulation. This introduces, however, a domain gap issue due to the difference in the distribution of data generated in simulation and the real-world scenarios. Recently, domain randomization technique \cite{53} has been utilized to address this issue for training pose estimation approaches \cite{47,54}. Additionally, approaches have been developed to explicitly bridge this domain gap between real and synthetic datasets \cite{55,56} via training techniques like Generative Adversarial Networks (GAN).

### 2.3. Point-set Registration Methods

Many recent pose estimation techniques \cite{1,57} integrate object segmentation with pointset registration techniques \cite{2}. Popular local registration approaches are Iterative Closest Points (ICP) \cite{58}) and its variants \cite{59,60,61,62,63}, which typically require a good initialization. Otherwise, registration requires finding the best aligning rigid transform over the 6-DOF space of all possible transforms, which are uniquely determined by 3 pairs of (non-degenerate) corresponding points. A popular strategy is to invoke \textsc{RANSAC} to find aligned triplets of point pairs \cite{64} but suffers from a frequently observable worst case $O(n^3)$ complexity in the number $n$ of data samples, which has motivated many extensions \cite{65,66}. The\textsc{4PCS} algorithm \cite{67} achieved $O(n^2)$ output-sensitive complexity using four congruent point basis instead of three. This method was extended to Super4PCS \cite{2}, which achieves $O(n)$ output-sensitive complexity. The accuracy of these methods, however, highly depends on the predictions from the object detector.

### 2.4. Global Scene-Level Reasoning

A popular approach to resolve conflicts arising from local reasoning is to generate object pose candidates and perform a Hypothesis Verification (HV) step \cite{68,69,70}. The hypotheses generation in most cases occurs using a variant of \textsc{RANSAC} \cite{21}. One of this method’s drawbacks
is that the generated hypotheses might already be conflicted due to errors in object segmentation and thus performing an optimization over this might not be very useful. Another approach to counter these drawbacks corresponds to an exhaustive but informed search to find the best scene hypotheses over a discrete set of object placement configurations \[71\]. A tree search formulation as described above was defined to effectively search in \(3-\text{DOF}\) space. Another recent work \[72\] performs the search for \(6-\text{DOF}\) pose by utilizing a particle filtering framework.

### 2.5. Object Representation for Robot Manipulation

Objects are often represented as mesh models that capture the surface of the object. The models are built either using a turntable setup \[73\], or via in-hand scanning by a human user \[74\] or a robotic arm \[75\]. A popular technique for surface reconstruction is Truncated Signed Distance Function (TSDF) \[76, 77\] which fuses multiple depth observations from a sensor and maintains a signed distance to the closest zero-crossing (representing the surface). Alternatively, the Surfel representation \[78\] is used to store local surface patches with position and normal information. The complete models are then used to perform pose estimation over the online sensor data and transfer the manipulation actions that are defined over the model to the scene. Given access to these object models, previous work has addressed problems such as bin-picking \[1\], tight-packing \[79\] and placement of grasped objects in clutter \[80\].

In certain scenarios, a model of the object is not known in advance. Most manipulation pipelines for novel objects \[81, 82\] focus on picking the object but have not addressed the problem of constrained placement. It has been demonstrated that robust grasps can be computed \[83\] over 3d point cloud representations of novel objects by learning local geometric features. However, constrained placement tasks require simultaneously evaluating placements and grasps over the objects, which is a relatively harder problem than task-agnostic grasping. A recent work, \[84\] performs pick-and-place of objects without object models by training an end-to-end deep reinforcement learning framework within the task context. Given that it is hard to interpret the learned policies, it is not clear how the policies learned with rewards coming from a specific task can be generalized to other similar tasks, configurations and objects. Other alternatives involve modeling the objects at category-level via a normalized object frame \[85\].
a canonical model [86] or semantic keypoints [87]. Such approaches are often appended with volumetric shape completion [88, 89, 90, 91] for collision checking. Shape completion could be tricky given large intra-class variance in object shapes and assumptions such as symmetry often leading to undesirable scenarios such as collisions.

The thesis leverages the recent progress in deep learning for object detection (in Chapter 3) and instance segmentation (Chapter 6) and proposes data generation as well as training techniques that allow the networks to be trained entirely in simulation and then used over real images. Chapter 4 extends point-set registration to operate over a soft segmentation output from CNNs. It combines the probabilistic output from a learned network with a geometric sampling-based algorithm for robust and efficient pose estimation. Chapter 5 presents a new algorithm for global scene-level reasoning that leverages physics simulation to sequentially generate and validate scene hypothesis in a Monte Carlo Tree Search. Chapter 6 proposes a learning-based framework for scene-level estimation that could handle multiple instances of densely packed objects in a computationally efficient manner. Finally, Chapter 8 presents a manipulation planning framework for picking and constrained placement of previously unseen objects. The planning is performed over a dynamic object representation that is continuously tracked and updated as the object is manipulated.
Chapter 3
Self-supervised Learning for Object Detection using Physics Simulation

Object detection and pose estimation is frequently the first step of robotic manipulation. Recently, deep learning methods, such as those employing Convolutional Neural Networks (CNNs), have become the standard tool for object detection, outperforming alternatives in object recognition benchmarks. These desirable results are typically obtained by training CNNs using datasets that involve a very large number of labeled images, as in the case of ImageNet [92]. Creating such large datasets requires intensive human labor. Furthermore, as these datasets are general-purpose, one needs to create new datasets for specific object categories and environmental setups that may be of importance to robotics, such as warehouse management and logistics.

The recent Amazon Picking Challenge (APC) [93] has reinforced this realization and has led into the development of datasets specifically for the detection of objects inside shelving units [94, 95, 1]. These datasets are created either with human annotation or by incrementally placing one object in the scene and using foreground masking.

An increasingly popular approach to avoid manual labeling is to use synthetic datasets generated by rendering 3D CAD models of objects with different viewpoints. Synthetic datasets have been used to train CNNs for object detection [96] and viewpoint estimation [97]. One major challenge in using synthetic data is the inherent difference between virtual training examples and real testing data. For this reason, there is considerable interest in studying the impact of texture, lighting, and shape to address this disparity [98]. Another issue with synthetic images generated from rendering engines is that they display objects in poses that are not necessarily physically realistic. Moreover, occlusions are usually treated in a rather naive manner, i.e., by applying cropping, or pasting rectangular patches, which again results in unrealistic
Figure 3.1: (Left) A robotic arm performs pose estimation from multiple viewpoints using an object detector trained with physically-simulated scenes (Top-right). The estimated poses are used to automatically label real images (Bottom-right). They are added to the training dataset as part of a lifelong learning process. Initially, the multi-view pose estimation procedure bootstraps its accuracy by trusting the objects’ labels as predicted from the detector given training over synthetic images. It then uses these labels to annotate images of the same scene taken from more complex viewpoints.

This chapter proposes an automated system for generating and labeling datasets for training CNNs. The objective of the proposed system is to reduce manual effort in generating data and to increase the accuracy of bounding-box-based object detection for robotic setups. In particular, the two main contributions of this work are:

- A physics-based simulation tool, which uses information from camera calibration, object models, shelf or table localization to setup an environment for generating training data. The tool performs physics simulation to place objects at realistic configurations and renders images of scenes to generate a synthetic dataset to train an object detector.

- A lifelong, self-learning process, which employs the object detector trained with the above physics-based simulation tool to perform a robust multi-view pose estimation with a robotic manipulator, and use the results to correctly label real images in all the different views. The key insight behind this system is the fact that the robot can often find a good viewing angle that allows the detector to accurately label the object and estimate its pose. The object’s predicted pose is then used to label images of the same scene taken from more difficult views, as shown in Fig. 3.1. The transformations between different views
Figure 3.2: Pipeline for physics aware simulation: The 3D CAD models are generated and loaded in a calibrated environment on the simulator. A subset of the objects is chosen for generating a scene. Objects are physically simulated until they settle on the resting surface under the effect of gravity. The scenes are rendered from known camera poses. Perspective projection is used to compute 2D bounding boxes for each object. The labeled scenes are used to train a Faster-RCNN object detector \([50]\), which is tested on a real-world setup.

are known because they are obtained by moving the robotic manipulator.

The software and data of the proposed system, in addition to all the experiments, are publicly available at [http://www.physimpose.com](http://www.physimpose.com)

### 3.1. Physics-aware Synthetic Data Generation

The first component of the proposed framework physically simulates scenes containing target objects and generates images of the corresponding scenes using the parameters of a known camera. This is used to generate a synthetic dataset for training a CNN-based object detector. The pipeline for this process is depicted in Figure 3.2.

The dataset generation process mimics a real-world setup involving a sensing system for robotic manipulation, where a camera is mounted on a robotic arm. The robot is placed in front of a surface for object placement (resting surface), such as a shelf-bin or table-top, which contains the objects. In such a setup, forward kinematics can be used to provide the 6-DoF pose \(T_{cam}\) of the camera. Furthermore, a camera calibration process provides the intrinsic parameters of the camera \(K\). The pose of the resting surface \(T_{rs}\) relative to the robot is determined by a RANSAC-based estimation process \([21]\). For instance, for the shelf depicted in Figure 3.2...
such a pose estimation process was implemented by computing the edges and planes on the retrieved depth data and matching them against the known geometry of the shelf.

Given the above information as input, the method aims to render and automatically label several images in simulation as discussed in Algorithm 1.

**Algorithm 1: PHYSIM\_CNN**(\(T_{cam}, T_{rs}, K, M_{1:N}\))

1. \(T_{cam}\): set of camera poses for rendering
2. \(T_{rs}\): pose of the resting surface
3. \(K\): intrinsic camera parameters
4. \(M_{1:N}\): mesh models for all \(N\) objects

1. \(\text{dataset} \leftarrow \emptyset;\)
2. \(\text{while } (|\text{dataset}| < \text{desired size}) \text{ do}\)
3. \(\quad \text{O} \leftarrow \text{a random subset of objects from } M_{1:N};\)
4. \(\quad T_{\text{init}}^{O} \leftarrow \text{INITIAL\_RANDOM\_POSES}(\text{O});\)
5. \(\quad T_{\text{final}}^{O} \leftarrow \text{PHYSICS\_SIM}(T_{\text{init}}^{O}, T_{rs}, \text{O});\)
6. \(\quad \text{Light} \leftarrow \text{PICK\_LIGHTING\_CONDITIONS}();\)
7. \(\text{foreach } (\text{view} \in T_{cam}) \text{ do}\)
8. \(\quad \text{image} \leftarrow \text{RENDER}(T_{\text{final}}^{O}, \text{view}, K, \text{Light});\)
9. \(\quad \{ \text{labels}, \text{bboxes} \} \leftarrow \text{PROJECT}(T_{\text{final}}^{O}, \text{view});\)
10. \(\quad \text{set of object poses } T_{\text{final}}^{O} \text{ is used to generate bounding-boxes in all views}\)
11. \(\text{dataset} \leftarrow \text{dataset} \cup (\text{image}, \text{labels}, \text{bboxes});\)
12. \(\text{Train FASTER\_RCNN with the generated dataset;}\)

The algorithm simulates a scene by first selecting randomly a set of objects \(O\) from the list of available object models \(M_{1:N}\) (line 3). The initial pose of an object is provided by function \text{INITIAL\_RANDOM\_POSES} (line 4), which samples uniformly at random along the \(x\) and \(y\)-axis from the range \((-\frac{\text{dim}_{i}}{2}, \frac{\text{dim}_{i}}{2})\), where \(\text{dim}_{i}\) is the dimension of the resting surface along the \(i^{th}\) axis. The initial position along the \(z\)-axis is fixed and can be adjusted to either simulate dropping or placing. The initial orientation is sampled appropriately in \(\text{SO}(3)\). Then, function \text{PHYSICS\_SIM} is called (line 5), which physically simulates the objects and allows them to fall due to gravity, bounce, and collide with each other as well as with the resting surface. Any inter-penetrations among objects or with the surface are treated by the physics engine. The final poses of the objects \(P_{\text{final}}^{O}\), when they stabilize, resemble real-world poses. Gravity, friction
coefficients, and mass parameters are set at similar values globally and damping parameters are set to the maximum to promote fast stabilization.

The environment lighting and point light sources are varied with respect to location, intensity, and color for each rendering (line 6). Simulating various indoor lighting sources helps to avoid over-fitting to a specific texture, which makes the training set more robust to different testing scenarios. Once lighting conditions are chosen, the simulated scene is rendered from multiple views using the pre-defined camera poses (line 6). The rendering function RENDER requires the set of stabilized object poses $T_{final}^O$, the camera viewpoint as well as the selected lighting conditions and intrinsic camera parameters (line 7). Finally, perspective projection is applied to obtain 2D bounding box labels for each object in the scene with function PROJECT (line 8). The overlapping portion of the bounding boxes for the object that is further away from the camera is pruned.

The generated synthetic dataset is used to train an object detector based on Faster-RCNN ([50]), which utilizes a deep VGG network architecture ([100]). The dataset generation module has been implemented using the Blender API ([101]), which internally uses the Bullet physics engine ([102]) and has been publicly shared.

A critical requirement for learning with synthetic data as discussed above is the need for modeling the domain in the simulation. The precision with which the geometry and texture of the objects and support surface need to be modeled depends on the set of objects to be detected. If the objects have very different geometries, a noisy modeling of the shape using surface reconstruction technique like KinectFusion ([77]) is good enough for the recognition task, such as in the Linemod dataset ([36]). If there are multiple objects with similar geometry, accuracy in modeling the texture and color is more critical to achieving a good performance, for example in the Shelf&Tote dataset ([1]). Other physical properties like mass and friction coefficients of objects have been kept as constant over all objects for the scope of this work while object material properties and parameters corresponding to the illumination of the environment have been randomized within a wide domain.

\footnote{Code: https://github.com/cmitash/physim-dataset-generator}
3.2. Self-Learning via Multi-view Pose Estimation

Given access to an object detector and a pose estimation process trained with the physics-based simulator, the self-learning pipeline labels real-world images with a robust multi-view pose estimation. This is based on the idea that the detector performs well on some views, while might be imprecise or fail in other views. Aggregating 3D data over the confident detections and with access to the knowledge of the environment, a 3D segment can be extracted for each object instance in the scene. This process, combined with the fact that 3D models of objects are available, makes it highly likely to estimate correct 6-DoF poses of objects given enough views and search time. The results of pose estimation are then projected back to the multiple views and used to label real images. These examples are very effective to reduce the confusion in the classifier for novel views. The process also autonomously reconfigures the scene using manipulation actions to apply the labeling process iteratively over time in different scenes, thus generating a labeled dataset which is used to re-train the object detector. The pipeline of the process is presented in Fig. 3.3 and the pseudocode is provided in Algo. 2.
Algorithm 2: SELF-LEARN(dataset, T_cam, M_{1:N})

// dataset: synthetic training dataset
// T_cam: set of camera poses to collect images
// M_{1:N}: mesh models for all N objects
1 while |dataset| < desired size do
  2 foreach view ∈ T_cam do
    3 {RGB\_view, D\_view} ← CAPTURE(view);
    // RGB-D images are collected by moving the camera to all views in T_cam
  4 foreach object O in the scene do
    5 Cloud\_O = \emptyset;
    6 foreach view ∈ T_cam do
      7 bbox ← SIM\_DETECT(RGB\_view);
      // bounding box is detected for object O in image RGB\_view
      8 if conf(bbox) > \varepsilon then
        9 Seg3d ← CONVERT3D(bbox, D\_view);
        10 Cloud\_O ← Cloud\_O ∪ Seg3d;
        // point cloud of object O is extracted from depth image according to bbox
    11 OUTLIER\_REMOVAL(Cloud\_O);
    12 T\_O ← COMPUTE6DPOSE(Cloud\_O, M\_O);
    // 6d pose is computed given the point cloud segment and object model
    13 foreach view ∈ T_cam do
      14 {labels, bboxs} ← PROJECT(T\_O, view);
      // Estimated pose T\_O is used to generate bounding-boxes in all views
      15 dataset ← dataset ∪ (RGB\_view, labels, bboxs);
      16 randObj ← SAMPLE\_RANDOM\_OBJECT(M_{1:N});
      17 RECONFIGURE\_OBJECT(randObj);
      // randomly selected objects are moved to pre-specified configuration.
  18 Train Faster-RCNN using the expanded dataset;

A robotic arm is used to move the sensor to different pre-defined camera configurations T_cam and capture RGB and depth images of the scene (lines 2-3). The PRACSYS motion planning library ([103], [104]) was used to control the robot in the accompanying implementation.

The detector trained using physics-aware simulation is then used to extract bounding boxes corresponding to each object in the scene (line 7). There might exist a bias in simulation either with respect to texture or poses, which can lead to imprecise bounding boxes or complete failure in certain views. For the detection to be considered for further processing, a threshold
is considered on the confidence value returned by \text{RCNN} (line 8).

The pixel-wise depth information Seg3d within the confidently detected bounding boxes bbox (line 9) is aggregated in a common point cloud per object \(Cloud_{O}\) given information from multiple views (line 10). The process employs environmental knowledge to clean the aggregated point cloud (line 11). Points outside the resting surface bounds are removed and outlier removal is performed based on k-nearest neighbors and a uniform grid filter.

Several point cloud registration methods were tested for registering the 3D model \(M_{O}\) with the corresponding segmented point cloud \(Cloud_{O}\) (line 12). This included \text{Super4PCS} ([2]), fast global registration ([105]) and simply using the principal component analysis (PCA) with Iterative Closest Point (ICP) ([58]). The \text{Super4PCS} algorithm used alongside ICP was found to be the most applicable for the target setup as it is the most robust to outliers and returns a very natural metric for confidence evaluation. \text{Super4PCS} returns the best rigid alignment according to the Largest Common Pointset (LCP). The algorithm searches for the best score, using transformations obtained from four-point congruences. Thus, given enough time, it generates the optimal alignment with respect to the extracted segment.

After the 6-DoF pose is computed for each object, the scene is recreated in the simulator using object models placed at the pose \(T_{O}\) and projected to the known camera views (line 14). Bounding boxes are computed on the simulated setup and transferred to the real images. This gives precise bounding box labels for real images in all the views (line 15).

To further reduce manual labeling effort, an autonomous scene reconfiguration is performed (lines 16-17). The robot reconfigures the scene with a pick and place manipulation action to iteratively construct new scenes and label them, as in Fig. 3.4. For each reconfiguration, the object to be moved is chosen randomly and the final configuration is selected from a set of pre-defined configurations in the workspace.

Finally, the \text{Faster-RCNN} network is re-trained with the expanded dataset. The factors
that prevent this process from a label drift are (1) The network is re-trained with a large number of accurate synthetic data. Thus, the training is immune to some amount of label noise in the self-labeled data. (2) Only the most confident detections from multiple-views are considered and a global search based process for pose estimation is used to obtain the estimates which are eventually used for labeling.

3.3. Evaluation

This section discusses the datasets considered, it compares different techniques for generating synthetic data and evaluates the effect of self-learning. It finally applies the trained detector on the 6DoF pose estimation task. The standard Intersection-Over-Union ($\text{IoU}$) metric is employed to evaluate performance in the object detection task.

3.3.1 Datasets

Several RGB-D datasets have been released in the setting of the Amazon Picking Challenge [94, 95, 1]. They proposed system was evaluated on the benchmark dataset released by Team MIT-Princeton called the Shelf&Tote dataset [1]. The experiments are performed on 148 scenes in the shelf environment with different lighting and clutter conditions. The scenes include 11 objects used in APC with 2220 images and 229 unique object poses. The objects were chosen to represent different geometric shapes but ignoring the ones which did not have enough depth information. Thus, the results can be generalized to a large set of objects.
The proposed system has been also evaluated on a real-world table-top setup. The corresponding test dataset was generated by placing multiple objects in different configurations on a table-top. An Intel RealSense camera mounted on a Motoman robot was used to capture images of scenes from multiple views. Images corresponding to 41 cluttered scenes, with 11 APC objects and 473 detection instances were collected and manually labeled.

### 3.3.2 Evaluating the Object Detector trained in Simulation

To study how object pose distribution effects the training process, different techniques for synthetic data generation are evaluated. The results of experiments performed on the Shelf&Tote dataset are presented in Table 3.1.

#### Generating training data using test data distribution

The objective here is to establish an upper bound for the performance of a detector trained with simulated images. For this purpose, the object detector is trained with the knowledge of pose distribution in the test data. This process consists of estimating the density of the test data with respect to object poses using *Kernel Density Estimation*, and generating training data according to this distribution, as follows:

- Uniformly simulate many scenes using a simulator and record the poses for each object in the scene.
- Weigh each generated scene according to its similarity to test data. This is the sum of the number of objects in the scene for which the pose matches (rotation difference less than $15^\circ$ and translation difference less than 5cm) at least one pose in their corresponding test pose distribution.
- Normalize the weights to get a probability distribution on the sampled poses.
- Sub-sample the training poses using the normalized probability distribution.

The sampled scenes were used to train a Faster-RCNN detector, which achieved an accuracy of 69%.
<table>
<thead>
<tr>
<th>Method</th>
<th>Success(IoU &gt; 0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team MIT-Princeton [1] (Benchmark)</td>
<td>75%</td>
</tr>
<tr>
<td>Sampled from test data distribution</td>
<td>69%</td>
</tr>
<tr>
<td>Sampled from uniform distribution</td>
<td>31%</td>
</tr>
<tr>
<td>Physics-aware simulation</td>
<td>64%</td>
</tr>
<tr>
<td>Physics-aware simulation + varying light</td>
<td>70%</td>
</tr>
<tr>
<td>Self-learning (2K images)</td>
<td>75%</td>
</tr>
<tr>
<td>Self-learning (6K images)</td>
<td>81%</td>
</tr>
<tr>
<td>Self-learning (10K images)</td>
<td>82%</td>
</tr>
</tbody>
</table>

Table 3.1: Evaluation on Princeton’s Shelf&Tote dataset [1]

**Uniformly sampled synthetic data**

This alternative is a popular technique of generating synthetic data. It uses 3D models of the objects to render their images from several viewpoints sampled on a spherical surface centered at the object. The background image corresponded to the APC shelf, on top of which randomly selected objects were pasted at sampled locations. This process allows to simulate occlusions and mask subtraction provides the accurate bounding boxes in these cases. The objects in these images are not guaranteed to have physically realistic poses. This method of synthetic data generation does not perform well on the target task, giving a low accuracy of 31%.

**Generating training data with physics-aware simulation**

The accuracy of 64% achieved by the proposed physics-aware simulator is close to the upper bound. By incorporating the knowledge of the camera pose, resting surface and by using physics simulation, the detector is essentially over-fitted to the distribution of poses from which the test data comes, which can be useful for robotic setups.

The results discussed until now were with respect to a constant lighting condition. As the dataset grows, then a dip in the performance is observed. This is expected as the detector overfits with respect to the synthetic texture, which does not mimic real lighting condition. This is not desirable, however. To deal with this issue, the lighting conditions are varied according to the location and color of the light source. This does resolve the problem to some extent but the dataset bias still limits performance to an accuracy of 70%. Fig. 3.5 shows examples of training images for different comparison methods.
On the table-top setup, the detector trained by the physics-based simulation has a success rate of 78.8%, as shown in Table 3.2.

<table>
<thead>
<tr>
<th>Method</th>
<th>Success(IoU &gt; 0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics aware simulation</td>
<td>78.8%</td>
</tr>
<tr>
<td>Self-learning (140 images)</td>
<td>90.3%</td>
</tr>
</tbody>
</table>

Table 3.2: Detection accuracy on table-top experiments

3.3.3 Evaluating Self-learning

The self-learning pipeline is executed over the training images in the Shelf&Tote training dataset to automatically label them using multi-view pose estimation. The real images are incrementally added to the simulated dataset to re-train the Faster-RCNN. This results in a performance boost of 12%. This result also outperforms the training process by which uses approximately 15,000 real images labeled using background subtraction. Training over clutter scenarios could be one of the critical reason that allows the proposed method to outperform a network trained with a large dataset of real images. Table 3.1 and Fig. 3.6 (Left) show the improvement from the self-learning process.

On the table-top setup, pose estimation is performed using the trained detector and model registration. The estimated poses with high confidence values are then projected to the known
camera views to obtain the 2D bounding box labels on real scenes. This is followed by re-configuration of the scenes using pick and place manipulation. After generating 140 scenes with a clutter of 4 objects in each image, the automatically labeled instances are used to retrain the Faster-RCNN detector. The performance improvement by adding these labeled examples is presented in Table 3.2. Fig. 3.6 (Right) shows an example of a new view learned via the self-learning process.

### 3.3.4 Evaluating the detector for 6DoF Pose estimation

Success in pose estimation is evaluated as the percentage of predictions with an error in translation less than 5cm and mean error in the rotation less than 15°. The results of pose estimation are compared to the pose system proposed by the APC Team MIT-Princeton [1] in addition to different model registration techniques. The results are depicted in Table 3.3. Given the above specified metric, the proposed approach outperforms the pose estimation system proposed before [1] by a margin of 25%. It is very interesting to note that the success in pose estimation task is at par with the success achieved using ground truth bounding boxes.

<table>
<thead>
<tr>
<th>2D-Segmentation Method</th>
<th>3D-registration Method</th>
<th>Rot. err. (°)</th>
<th>Trans. err. (m)</th>
<th>Success(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground-Truth Bounding-Box</td>
<td>PCA + ICP</td>
<td>7.65</td>
<td>0.02</td>
<td>84.8</td>
</tr>
<tr>
<td>FCN (trained with [1])</td>
<td>PCA + ICP</td>
<td>17.3</td>
<td>0.06</td>
<td>54.6</td>
</tr>
<tr>
<td>FCN (trained with [1])</td>
<td>Super4PCS + ICP</td>
<td>16.8</td>
<td>0.06</td>
<td>54.2</td>
</tr>
<tr>
<td>FCN (trained with [1])</td>
<td>fast-global-registration</td>
<td>18.9</td>
<td>0.07</td>
<td>43.7</td>
</tr>
<tr>
<td>RCNN (Proposed training)</td>
<td>PCA + ICP</td>
<td>8.50</td>
<td>0.03</td>
<td>79.4</td>
</tr>
<tr>
<td>RCNN (Proposed training)</td>
<td>Super4PCS + ICP</td>
<td>8.89</td>
<td>0.02</td>
<td>75.0</td>
</tr>
<tr>
<td>RCNN (Proposed training)</td>
<td>fast-global-registration</td>
<td>14.4</td>
<td>0.03</td>
<td>58.9</td>
</tr>
</tbody>
</table>

Table 3.3: Comparing the performance of the proposed system to state-of-the-art techniques for pose estimation.
Chapter 4

Robust Pose Estimation with Stochastic Congruent Sets

As discussed in the previous chapter, pose estimation often involves two sub-components, image-based object recognition and searching in SE(3) to estimate a unique pose for the target object. Many recent approaches [1, 57, 51, 50] treat object segmentation by using a Convolutional Neural Network (CNN), which provides either a bounding-box detection or a per-pixel classification. In either case, a hard object segment is computed that often includes noise such as under-segmentation or over-segmentation, as shown in Fig. 4.1. Such noise arise especially when the CNNs is only trained on synthetic data and applied over real images.

Segmentation is followed by a 3D model alignment using point cloud registration, such as ICP [58], or global search alternatives, such as 4-points congruent sets (4-PCS) [67, 2]. These methods operate over two deterministic point sets \( S \) and \( M \). They sample iteratively, a base \( B \) of 4 coplanar points on \( S \) and try to find a set of 4 congruent points on \( M \), given geometric constraints, so as to identify a relative transform between \( S \) and \( M \) that gives the best alignment score. The pose estimate from such a process is incorrect when the segment is noisy or if it does not contain enough points from the object.

This chapter is based on the contributed publication [7] that treats the CNN output as a probability for an object to be visible at each pixel. These segmentation probabilities can then be used during the registration process to achieve robust and fast pose estimation. This requires sampling a base \( B \) on a segment, such that all points on the base belong to the target object with high probability. The resulting approach, denoted as “Stochastic Congruent Sets” (StoCS), achieves this by building a probabilistic graphical model given the obtained soft segmentation and information from the pre-processed geometric object models. The pre-processing corresponds to building a global model descriptor that expresses oriented point pair features [30]. This geometric modeling, not only biases the base samples to lie within the object bound, but
Figure 4.1: (a) A robotic arm using pose estimates from StoCS to perform manipulation. (b) Hard segmentation errors adversely affect model registration. (c) Heatmaps showing the continuous probability distribution for an object. (d) Pose estimates obtained by StoCS.

is also used to constrain the search for finding the congruent sets, which provides a substantial computational benefit.

Thus, this chapter presents two key insights: 1) it is not necessary to make hard segmentation decisions prior to registration, instead the pose estimation can operate over the continuous segmentation confidence output of CNNs. 2) Combining a global geometric descriptor with the soft segmentation output of CNNs intrinsically improves object segmentation during registration without a computational overhead.

StoCS is first tested on a dataset of cluttered real-world scenes by using the output of an FCN that was trained solely on a synthetic dataset. In such cases, the resulting segmentation is quite noisy. Nevertheless, experiments show that high accuracy in pose estimation can be achieved with StoCS. The method has also been evaluated on the YCB object dataset [48], a benchmark for robotic manipulation, where it outperforms modern pointset registration and pose estimation techniques in accuracy. It is much faster than competing registration processes, and only slightly slower than end-to-end learning.

4.1. Problem statement

Consider the problem of estimating the 6D poses of $N$ known objects $\{O_1, \ldots, O_N\}$, captured by an RGB-D camera in an image $I$, given their 3D models $\{M_1, \ldots, M_N\}$. The estimated
poses are returned as a set of rigid-body transformations \( \{T_1, \ldots, T_N\} \), where each \( T_i = (t_i, R_i) \) captures the translation \( t_i \in \mathbb{R}^3 \) and rotation \( R_i \in SO(3) \) of object model \( \mathcal{M}_i \) in the camera’s reference frame. Each model is represented as a set of 3D surface points sampled from the object’s CAD model by using Poisson-disc sampling.

4.2. Defining the Segmentation-based Prior

The proposed approach uses as prior the output of pixel-wise classification. For this purpose, a fully-convolutional neural network [51] is trained for semantic segmentation using RGB images annotated with ground-truth object classes. The learned weights of the final layer of the network \( w_k \) are used to compute \( \pi(p_i \rightarrow O_k) \), i.e., the probability pixel \( p_i \) corresponds to object class \( O_k \). In particular, this probability is defined as the ratio of the weight \( w_k[p_i] \) over the sum of weights for the same class over all pixels \( p \) in the image \( I \):

\[
\pi(p_i \rightarrow O_k) = \frac{w_k[p_i]}{\sum_{p \in I} w_k[p]}.
\]

These pixel probabilities are used to construct a point cloud segment \( S_k \) for each object \( O_k \) by liberally accepting pixels in the image that have a probability greater than a positive threshold \( \varepsilon \) and projecting them to the 3D frame of the camera. The segment \( S_k \) is accompanied by a probability distribution \( \pi_k \) for all the points \( p \in S_k \), which is defined as follows:

\[
S_k \leftarrow \{p_i \mid p_i \in I \wedge \pi(p_i \rightarrow O_k) > \varepsilon\},
\]

\[
\pi_k(p) = \frac{\pi(p_i \rightarrow O_k)}{\sum_{q \in S_k} \pi(q \rightarrow O_k)}.
\]

Theoretically, \( \varepsilon \) can be set to 0, thus considering the entire image. In practice, \( \varepsilon \) is set to a small value to avoid areas that have minimal probability of belonging to the object.

4.3. Congruent Set Approach for Computing the Best Transform

The objective reduces to finding the rigid transformation that optimally aligns the model \( \mathcal{M}_k \) given the point cloud segment \( S_k \) and the accompanying probability distribution \( \pi_k \). To account
Algorithm 3: StoCS($S_k$, $\pi_k$, $M_k$)

1. bestScore ← 0;
2. $T_{opt}$ ← identity transform;
3. while runtime < max runtime do
4.   $B$ ← SELECT_StoCS_BASE($S_k$, $\pi_k$, $M_k$);
5.   $U$ ← FIND_CONGRUENT_SETS($B$, $M_k$);
6.   foreach 4-point set $U_j \in U$ do
7.     $T$ ← best rigid transform that aligns $B$ to $U_j$ in the least squares sense;
8.     score ← $\sum_{m_i \in M_k} f(m_i, T, S_k)$;
9.     if score > bestScore then
10.       bestScore ← score; $T_{opt}$ ← $T$;
11. return $T_{opt}$;

for the noise in the extracted segment and the unknown overlap between the two pointsets, the alignment objective $T_{opt}$ is defined as the matching between the observed segment $S_k$ and the transformed model, weighted by the probabilities of the pixels. In particular:

$$T_{opt} = \arg\max_T \sum_{m_i \in M_k} f(m_i, T, S_k),$$

where $s^*$ is the closest point on segment $S_k$ to model point $m_i$ after $m_i$ is transformed by $T$; $N(.)$ is the surface normal at that point; $\delta_d$ is the acceptable distance threshold and $\delta_n$ is the surface normal alignment threshold. Algorithm 3 explains how to find $T_{opt}$.

The proposed method follows the principles of randomized alignment techniques and at each iteration samples a base $B$, which is a small set of points on the segment $S_k$. The sampling process also takes into account the probability distribution $\pi_k$ as well as geometric information regarding the model $M_k$. To define a unique rigid transform $T_i$, the cardinality of the base should be at least three. Nevertheless, inspired by similar methods [67, 2], the accompanying implementation samples four points to define a base $B$ for increased robustness. The following section details the base selection process.

For the sampled base $B$, a set $U$ of all similar or congruent 4-point sets is computed on the model point-set $M_k$, i.e., $U$ is a set of tuples with 4 elements. For each of the 4-point sets $U_j \in U$
The first point of the base is sampled from the prior distribution obtained from a CNN.

\[ \pi(p|b_1) \]

The edge factor \( \phi_{\text{edge}}(p, b_1) \) is computed for each point \( p \) on the segment \( S \) based on the presence of similar features on the object model.

Sampled base \( B = \{b_1, b_2, b_3, b_4\} \)

The probability distribution is updated based on the already sampled points on the base.

The probability of \( p \) depends on the existence of \( \phi_{\text{edge}}(p, b_1), \phi_{\text{edge}}(p, b_2), \phi_{\text{edge}}(p, b_3) \) on the object model.

Figure 4.2: A description of the stochastic optimization process for extracting the base \( B = \{b_1, b_2, b_3, b_4\} \) so that it is distributed according to the stochastic segmentation and in accordance with the object’s known geometry. The base is matched against candidate sets \( U = \{U_1, \ldots, U_N\} \) of 4 congruent points each from the object model \( M \).

The method computes a rigid transformation \( T \), for which the optimization cost is evaluated, and keeps track of the optimum transformation \( T_{\text{opt}} \). In the general case, the stopping criterion is a large number of iterations, which are required to ensure a minimum success probability with randomly sampled bases. In practice, however, the approach stops after a maximum predefined runtime is reached.

### 4.4. Stochastic Optimization for Selecting the Base

The process for selecting the base is given in Alg. 4 and highlighted in Fig. 4.2. As only a limited number of bases can be evaluated in a given time frame, it is critical to ensure that all base points belong to the object in consideration with high probability. Using the Hammersley-Clifford factorization, the joint probability of points in \( B = \{b_1, b_2, b_3, b_4\} \) belonging to \( O_k \) is given as:

\[
Pr(B \rightarrow O_k) = \frac{1}{Z} \prod_{i=0}^{m} \phi(C_i),
\]

(4.4)

where \( C_i \) is defined as a clique in a fully-connected graph that has as nodes \( b_1, b_2, b_3 \) and \( b_4 \). \( Z \) is the normalization constant and \( \phi(C_i) \) corresponds to the factor potential of the clique \( C_i \). For computational efficiency, only cliques of sizes 1 and 2 are considered, which are respectively the nodes and edges in the complete graph of \( \{b_1, b_2, b_3, b_4\} \). The above simplification gives rise to the following approximation of Eqn. 4.4:
\[
Pr(B \rightarrow O_k) = \frac{1}{Z} \prod_{i=1}^{4} \{\phi_{\text{node}}(b_i) \prod_{j=1}^{j<i} \phi_{\text{edge}}(b_i, b_j)\}.
\]

The above operation is implemented efficiently in an incremental manner. The last element of the implementation is the definition of the factor potentials for nodes and edges of the graph \(\{b_1, b_2, b_3, b_4\}\). The factor potential for the nodes can be computed by using the class probabilities returned by the CNN-based soft segmentation, i.e.

\[
\phi_{\text{node}}(b_i) = \pi_k(b_i).
\]

The factor potentials \(\phi_{\text{edge}}\) for edges can be computed using the Point-Pair Feature (PPF) of the two points defining the edge and the frequency of the computed feature on the CAD model \(M_i\) of the object. The PPF for two points on the model \(m_1, m_2\) with surface normals \(n_1, n_2\):

\[
\text{PPF}(m_1, m_2) = (||d||, \angle(n_1, d), \angle(n_2, d), \angle(n_1, n_2)),
\]

wherein \(d = m_2 - m_1\) is the vector from \(m_1\) to \(m_2\).

A hash map is generated for the object model, which counts the number of occurrences of discretized point pair features in the model. To account for the sensor noise, the point pair features are discretized. Nevertheless, even with discretization, the surface normals of points in the scene point cloud could be noisy enough such that they do not map to the same bin as the corresponding points on the model. To overcome this issue, during the model generation process, each point pair also votes to several neighboring bins. For the accompanying implementation, the bin discretization was kept at 10 degrees and 0.5 cm. The point-pair features voted to \(2^4\) other bins in the neighborhood of the bin the feature points to. This ensures the robustness of the method in case of noisy surface normal computations. Then, the factor potential for edges in the base is given as:
Algorithm 4: SELECT_StoCS_BASE \((S_k, \pi_k, M_k)\)

1. \(b_1 \leftarrow \text{sample a point from } S_k \text{ according to the discrete probability distribution defined by} \pi_k\); 
2. \(\text{foreach point } p \in S_k \text{ do}\)
   3. \(\pi(p|b_1) = \pi_k(p)\pi_k(b_1)\phi_{edge}(p, b_1)\); 
   4. \(b_2 \leftarrow \text{sample from normalized } \pi(.|b_1)\); 
5. \(\text{foreach point } p \in S_k \text{ do}\)
   6. \(\pi(p|b_1, b_2) = \pi(p|b_1)\pi(b_2|b_1)\phi_{edge}(p, b_2)\); 
   7. \(\text{if } \angle((p - b_0), (b_1 - b_0)) < \varepsilon_1 \text{ then}\)
      8. \(\pi(p|b_1, b_2) \leftarrow 0;\) 
   9. \(b_3 \leftarrow \text{sample from normalized } \pi(.|b_1, b_2)\); 
10. \(\text{foreach point } p \in S_k \text{ do}\)
    11. \(\pi(p|b_1, b_2, b_3) = \pi(p|b_1, b_2)\pi(b_3|b_1, b_2)\phi_{edge}(p, b_3)\); 
    12. \(\text{if } \text{distance}(\text{plane}(b_1, b_2, b_3), p) < \varepsilon_2 \text{ then}\)
        13. \(\pi(p|b_1, b_2, b_3) \leftarrow 0;\) 
    14. \(b_4 \leftarrow \text{sample from normalized } \pi(.|b_1, b_2, b_3)\); 
15. \(\text{return } b_1, b_2, b_3, b_4;\)

\[
\phi_{edge}(b_i, b_j) = \begin{cases} 
1, & \text{if } \text{hashmap}(M_k, \text{PPF}(b_i, b_j)) > 0 \\
0, & \text{otherwise}
\end{cases}
\]

Thus, the sampling of bases incorporates the above definitions and proceeds as described in Algorithm 4. In particular, each of the four points in a base \(B\) is sampled from the discrete probability distribution \(\pi_k\), defined for the point segment \(S_k\). This distribution is initialized as shown in Eqns. 4.1 and 4.3 using the output of the last layer of a CNN. The probability of sampling a point \(p \in S_k\) is incrementally updated in Algorithm 4 by considering the edge potentials of points with already sampled points in the base. This step essentially prunes points that do not relate, according to the geometric model of the object, to the already sampled points in the base. Furthermore, constraints are defined in the form of conservative thresholds \((\varepsilon_1, \varepsilon_2)\) to ensure that the selected base has a wide interior angle and is coplanar.

The \textsc{FindCongruentSets}(B, M_k) subroutine of Algorithm 3 is used to compute a set \(\mathcal{U}\) of 4-points from \(M_k\) that are congruent to the sampled base \(B\). The 4-points of the base can be represented by two pairs represented by their respective PPF and the ratio defined on the line segments by virtue of their intersection. Two sets of point pairs are computed on the model
with the PPFs specified by the segment base. The pairs in the two sets, which also intersect with the given ratios are classified as congruent 4-points. The basic idea of 4 point congruent sets was originally proposed in [67]. It was derived from the fact that these ratios and distances are invariant across any rigid transformation. In StoCS the pairs are compared using point-pair features instead of just distances, which further reduces the cardinality of the sets of pairs that need to be compared and thus speed-ups the search process.

4.5. Evaluation

Two different datasets are used for the evaluation of the proposed method.

4.5.1 Amazon Picking Challenge (APC) dataset

This RGB-D dataset [8] contains real images of multiple objects from the Amazon Picking Challenge (APC) in varying configurations involving occlusions and texture-less objects.

A Fully Convolutional Network (FCN) was trained for semantic segmentation by using synthetic data. The synthetic images were generated by a toolbox for this dataset [6]. A dataset bias was observed, leading to performance drop on mean recall for pixel-wise prediction from 95.3% on synthetic test set to 77.9% on real images. Recall can be improved by using a continuous probability output from the FCN with no or very low confidence threshold as proposed in this work. This comes at the cost of losing precision and including parts of other objects in the segment being considered for model registration. Nevertheless, it is crucial to achieve accurate pose estimation on real images given a segmentation process trained only on synthetic data as it significantly reduces labeling effort.

Table 4.1 provides the pose accuracy of StoCS compared against Super4PCS and V4PCS. The Volumetric-4PCS (V4PCS) approach samples 4 base points by optimizing for maximum volume and thus coplanarity is no more a constraint. Congruency is established when all the edges of the tetrahedron formed by connecting the points have the same length. The performance is evaluated as mean error in translation and rotation, where the rotation error is a mean of the roll, pitch, and yaw error. The three processes sample 100 segment bases and verify all the transformations extracted from the congruent sets. While StoCS uses soft segmentation
<table>
<thead>
<tr>
<th>Method</th>
<th>Rot. error</th>
<th>Tr. error</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super4PCS [2]</td>
<td>8.83°</td>
<td>1.36cm</td>
<td>28.01s</td>
</tr>
<tr>
<td>V4PCS [106]</td>
<td>10.75°</td>
<td>5.48cm</td>
<td>4.66s</td>
</tr>
<tr>
<td>StoCS (OURS)</td>
<td>6.29°</td>
<td>1.11cm</td>
<td>0.72s</td>
</tr>
</tbody>
</table>

Table 4.1: Average rotation error, translation error and execution time (per object) over APC dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Base Sampling</th>
<th>Set Extraction</th>
<th>Set Verification</th>
<th>#Set per base</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super4PCS [2]</td>
<td>0.0045s</td>
<td>2.43s</td>
<td>19.98s</td>
<td>1957.18</td>
</tr>
<tr>
<td>V4PCS [106]</td>
<td>0.0048s</td>
<td>1.98s</td>
<td>0.36s</td>
<td>46.61</td>
</tr>
<tr>
<td>StoCS (OURS)</td>
<td>0.0368s</td>
<td>0.27s</td>
<td>0.37s</td>
<td>53.52</td>
</tr>
</tbody>
</table>

Table 4.2: Computation complexity for the different components of the registration process.

output, the segment for the competing approaches was obtained by thresholding on per-pixel class prediction probability. In Table 4.1, the optimal value of the threshold ($\varepsilon = 0.4$) is used for Super4PCS and V4PCS. In Figure 4.3, the robustness of all approaches is validated for different thresholds. The percentage of successful estimates (error less than 2cm and 10 degrees) reduces with the segmentation accuracy for both Super4PCS and V4PCS. But StoCS provides robust estimates even when the segmentation precision is very low. The StoCS output using FCN segmentation is comparable to results with registration on ground-truth segmentation, which is an ideal case for the alternative methods. This is important as it is not always trivial to compute the optimal threshold for a test scenario.

4.5.2 Computational cost

The computational cost of the process can be broken down into 3 components: base sampling, congruent set extraction, and set verification. StoCS increases the cost of base sampling as it iterates over the segment to update probabilities. But this is linear in the size of the segment and is not the dominating factor in the overall cost. The congruent set extraction and thus the verification step are output sensitive as the cost depends on the number of matching pairs on the model corresponding to 2 line segments on the sampled base for Super4PCS and StoCS and 6 line segments of the tetrahedron for V4PCS. Thus, base sampling optimizes for wide interior angle or large volume in Super4PCS and V4PCS respectively to reduce the number of similar sets on the model. This optimization, however, could lead to the selection of outlier points...
Figure 4.3: (Left) Robustness with varying segmentation confidences (Right) Anytime results for 3 pointset registration methods

in the sampled base, which occurs predominantly in V4PCS. For Super4PCS the number of congruent pairs still turns out to be very large (approx., 2000 per base), thus leading to a computationally expensive set extraction and verification stage. This is mostly seen for objects with large surfaces and symmetric objects. StoCS can restrict the number of congruent sets by only considering pairs on the model, which have the same PPF as on the sampled base. It does not optimize for wide interior angle or maximizing volume, but imposes a small threshold, such that nearby points and redundant structures are avoided in base sampling. So it can handle the computational cost without hurting accuracy as shown in Table 4.2 part (c).

4.5.3 YCB-Video dataset

The YCB-Video dataset [48] is a benchmark for robotic manipulation tasks that provides scenes with a clutter of 21 YCB objects [73] captured from multiple views and annotated with 6-DOF object poses. Along with the dataset, the authors also proposed an approach, PoseCNN, which learns to predict the object center and rotation solely on RGB images. The poses are further fine-tuned by initializing a modified ICP with the output of PoseCNN, and applying it on the depth images. The metric used for pose evaluation in this benchmark measures the average distance between model points transformed using the ground truth transformation and with the predicted transform. An accuracy-threshold curve is plotted and the area under the curve is reported as a scalar representation of the accuracy for each approach. To ignore errors caused due to object symmetry, the closest symmetric point is considered as a correspondence
Table 4.3: Success given the area under the accuracy-threshold curve and computation time (per object) on the YCB-Video dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Pose success</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PoseCNN [48]</td>
<td>57.37%</td>
<td>0.2s</td>
</tr>
<tr>
<td>PoseCNN+ICP [48]</td>
<td>76.53%</td>
<td>10.6s</td>
</tr>
<tr>
<td>PPF-Hough [30]</td>
<td>83.97%</td>
<td>7.18s</td>
</tr>
<tr>
<td>Super4PCS [2]</td>
<td>87.21%</td>
<td>43s</td>
</tr>
<tr>
<td>V4PCS [106]</td>
<td>77.34%</td>
<td>4.32s</td>
</tr>
<tr>
<td>StoCS (OURS)</td>
<td>90.1%</td>
<td>0.59s</td>
</tr>
</tbody>
</table>

The results of the evaluation are presented in Table 4.3. The accuracy of PoseCNN is low, mostly because it does not use depth information. When combined with a modified ICP, the accuracy increases but at a cost of large computation time. The modified ICP performs a gradient-descent in the depth image by generating a rendering score for hypothesized poses. The results are reported by running the publicly shared code separately over each view of the scene, which may not be optimal for the approach but is a fair comparison point as all the compared methods are tested on the same data and with the same computational resources.

For evaluating the other approaches, the same dataset used to train PoseCNN was employed to train FCN for semantic segmentation with a VGG16 architecture. A deterministic segment was computed based on thresholding over the network output. An alternative that is evaluated is Hough voting [30]. This achieves better accuracy compared to PoseCNN but is computationally expensive. This is primarily due to the quadratic complexity over the points on the segment, which perform the voting. Next, alternative congruent set based approaches were evaluated, Super4PCS and V4PCS. For each approach 100 iterations of the algorithm were executed. As the training dataset was similar to the test dataset, and an optimal threshold was used, 100 iterations were enough for Super4PCS to find good pose estimates. Nevertheless, Super4PCS generates a large number of congruent sets, even when surface normals were used to prune correspondences, leading to large computation time. V4PCS achieves lower accuracy. During its base sampling process, V4PCS optimizes for maximizing volume, which often biases towards outliers.

Finally, the proposed approach was tested. A continuous soft segmentation output was used
in this case instead of optimal threshold and 100 iterations of the algorithm was run. It achieves the best accuracy, and the computation time is just slightly larger than PoseCNN which was designed for time efficiency as it uses one forward pass over the neural network.
Chapter 5

Joint Pose Estimation in Clutter via Physics-aware Monte Carlo Tree Search

This chapter goes a step beyond the general paradigm of using a Convolutional Neural Network (CNN) for object segmentation [1, 57] followed by point cloud registration for pose estimation [2, 58]. The focus of this chapter [8, 5] is to improve the accuracy of pose estimation by reasoning at a scene-level about the physical interactions between objects.

In particular, existing object-level reasoning for pose estimation can fail on several instances. One reason can be imperfect object detection, which might include parts of other objects, thus guiding the model registration process to optimize for maximum overlap with a mis-segmented object point cloud. Another challenge is the ambiguity that arises when the object is only partially visible, due to occlusion. This results in multiple model placements with similar alignment scores, and thus a failure in producing a unique, accurate pose estimate. These issues were the primary motivation behind hypothesis verification (HV) methods [68, 69]. These techniques follow a pipeline where: (a) they first generate multiple candidate poses per object given feature matching against a model, (b) they create a set of scene-level hypotheses by considering the Cartesian product of individual object pose candidates, and find the optimal hypothesis with respect to a score defined in terms of similarity with the input scene and geometric constraints.

It has been argued, however, that existing HV methods suffer from critical limitations for pose estimation [107, 71]. The argument is that the optimization in the HV process, may not work because the true poses of the objects may not be included in the set of generated candidates, due to errors in the process for generating these hypotheses. Errors may arise from the fact that the training for detection typically takes place for individual objects and is not able to handle occlusions in the case of cluttered scenes.
This motivated the development of a search method [107, 71] for the best explanation of the observed scene by performing an exhaustive but informed search through rendering all possible scene configurations given a discretization over 3-DOFs, namely $(x, y, yaw)$. The search was formulated as a tree, where nodes corresponded to a subset of objects placed at certain configurations and edges corresponded to adding one more object in the scene. The edge cost was computed based on the similarity of the input image and the rendered scene for the additional object. An order of object placements over the tree depth was implicitly defined by constraining the child state to never occlude any object in the parent state. This ensured an additive cost as the search progressed, which allows the use of heuristic search.

This proposed technique adapts this idea of tree search to achieve better scalability and increased accuracy, while performing a comprehensive hypothesis verification process by addressing the limitations of such HV techniques. In particular, instead of imposing a discretization, which is difficult to scale to 6-DOF problems, the proposed search is performed over scene hypotheses. In order to address the issue of potentially conflicting candidate object poses, the scene hypotheses are dynamically constructed by introducing a constrained local optimization step over candidate object poses returned by Super4PCS, a fast global model matching method [2]. To limit detection errors that arise in cluttered scene, the proposed method builds on top of a previous contribution [6], which performs clutter-specific autonomous training to get object segments. This chapter provides experimental indications that the set of candidate object poses returned by Super4PCS given the clutter-aware training contains object poses that are close
enough to the ground truth, however, these might not be the ones that receive the best matching score according to Super4PCS. This is why it becomes necessary to search over the set of candidate poses. Searching over all possible hypotheses returned by Super4PCS, however, is impractical. Thus, this work introduces a clustering approach that identifies a small set of candidate pose representatives that is also diverse enough to express the spread of guesses in the matching process.

The search operates by picking candidate object poses from the set of cluster representatives given a specific order of object placement. This order is defined by considering the dependencies among objects, such as, when an object is stacked on top of another or is occluded by another object. At every expansion of a new node, the method uses the previously placed objects to re-segment the object point cloud. It then performs local point cloud registration as well as physics simulation to place the object in physically consistent poses with respect to the already placed objects and the resting surface. As the ordering considers the physical dependency between objects, the rendering cost is no longer additive and cannot be defined for intermediate nodes of the search tree. For this reason, a Monte Carlo Tree Search (MCTS) approach is used to heuristically guide the search based on evaluation of the rendering cost for the complete assignment of object poses. Specifically, the UCT search method is used with the Upper Confidence Bound (UCB) to trade-off exploration and exploitation in the search tree.

The experimental evaluation of the proposed framework indicates that searching over the space of scene hypotheses in this manner can quickly identify physically realistic poses that are close to ground truth and can significantly improve the quality of the output of global model matching methods.

5.1. Problem Setup

This work considers the problem of estimating the 6D poses of \( N \) known objects \( \{O_1, \ldots, O_N\} \) in a scene, captured by an RGB-D camera. The knowledge of the following elements is assumed:

- geometric models are given as textured triangular meshes \( \{1, \ldots, N\} \) of all the objects that are present in the scene. Mass of objects are kept as constant across all objects and
friction as well as linear and angular damping coefficients for objects are set to maximum within the simulator.

- triangular mesh and pose $T_{rs}$ for the resting surface of the objects, such as a shelf or a table in a global reference frame,

- the intrinsic and extrinsic parameters $K, T_{cam}$ for the camera.

The estimated poses are returned as a set of rigid-body transformations $\{T_1, \ldots, T_N\}$, where each $T_i = (t_i, R_i)$ captures the translation $t_i \in \mathbb{R}^3$ and rotation $R_i \in SO(3)$ of object model $i$ in a globally defined reference frame.

The proposed method approaches the problem by (1) generating a set of pose hypotheses for each object present in the scene, and (2) searching efficiently over the set of joint hypotheses for the most globally consistent solution. Global consistency is quantitatively evaluated by a score function. The score function measures the similarity between the actual observed depth image and a rendering of the objects in simulation using their hypothesized poses. The hypothesized poses are adapted during the search process, so as to correspond to poses where the objects are placed in a physically realistic and stable configuration according to a physics engine that simulates rigid object dynamics.

### 5.2. Hypothesis Generation

Some of the desired properties for a set of 6D pose hypotheses are the following:

- informed and diverse enough such that the optimal solution is either already contained in the set or a close enough hypothesis exists so that a local optimization process can fine-tune it and return a good result;

- limited in size, as evaluating the dependencies among the hypotheses set for different objects can lead to a combinatorial explosion of possible joint poses and significantly impact the computational efficiency;

- does not require extensive training.
The image describes the process of hypotheses generation for objects present in the scene. The process starts with extracting object segments \( S_{1:3} \) using Faster-RCNN [50], followed by using a global point cloud registration technique [2] to compute a set of possible model transformations \( (T_{1:3}) \) that corresponds to the respective segments. These transformations are then clustered to produce object specific hypotheses sets \( (H_{1:3}) \).

This work considers all of these properties while generating the hypothesis set. The pseudocode for hypothesis generation is presented in Algorithm 5.

The detector trained with the autonomous training process proposed in the previous section is used to extract bounding-box \( (bbox_O) \) for each object \( O \) in the scene. This, in turn, gives a segment \( P_O \) of the 3D point cloud. Segment \( P_O \) is a subset of the point cloud of the scene and contains points from the visible part of the object \( O \). Segment \( P_O \) frequently contains some points from nearby objects because the bounding box does not perfectly match the shape of the object.

The received point-set \( P_O \) is then matched to the object model \( M_O \) in the subroutine congruent set matching in Algorithm 5 to generate pose candidates for the object. This module, inspired by the Super4PCS [2] algorithm iteratively samples a set of 4 co-planar points from \( P_O \) called the base and finds sets of 4-points on the model which are congruent under rigid transformation, to the base. Each pair of congruent sets gives a pose hypothesis. The matching process is depicted in Fig. 5.3. The fact that the distances, angles, and ratios of the intersection of line segments are maintained over a rigid transform is used to come up with an efficient linear time algorithm for finding the congruent sets. The time complexity of this process is \( O(n + m + k) \), where \( n \) is the number of points on the sampled object model, \( m \) is the number of point-pairs on the model which are at the same distance as a point-pair on the sampled base and \( k \) is the number of the congruent sets found corresponding to the sampled base. As opposed
Algorithm 5: GEN_HYPOTHESIS(RGB, depth, M_{1:N})

// Given an RGB-D image and a set of object models M_{1:N},
// GEN_HYPOTHESIS generates pose candidates h_O for each
// object O.
1  H ← \{h_O = \emptyset, \forall O \in M_{1:N}\};
2  foreach object O in the scene do
3     bbox_O ← RCNN_DETECT(RGB, O);
4     P_O ← GET_3DPOINTS(bbox_O, depth);
5     T_O ← CONGRUENT_SET_MATCHING(M_O, P_O);
6     \{cluster_{tr}, center_{tr}\} ← KMEANS_{tr}(T_O);
7     foreach cluster C in cluster_{tr} do
8         \{cluster_{rot}, center_{rot}\} ← K-KMEANS_{tr}(C);
9         h_O ← h_O ∪ (center_{tr}, center_{rot});
10        H ← H ∪ h_O;
11  return H;

to RANSAC (21) which has a time complexity of \(O(n^3)\) for matching a set of 3 points to all
triplets of points on the model, this process better exploits the geometric constraints from rigid
transformations and efficiently produces a relatively small set of pose candidates.

Nevertheless, Super4PCS evaluates each of these transformations to find the one that
achieves the best alignment according to the LCP (Largest Common Pointset) metric. This
returned transformation, however, is not necessarily the optimal object pose as the point cloud
segment extracted via the detection process could include parts of other objects or due to lack of
visible surface might not be informative enough to compute the correct solution. This motivates
the consideration of other possible transformations for the objects, which can be evaluated in
terms of scene-level consistency.

Thus, the proposed process retains a set of possible transformations T_O computed using
congruent set matching within a given time budget t_o. It is interesting to consider the quality of
Figure 5.3: Figure depicts the congruent-set-matching process which finds sets of 4-points on the scene and on the object model that are congruent under rigid transformation. All the point-pairs on the model M with distances similar to $|p_1 - p_3|$ and $|p_2 - p_4|$ on the sampled base, can be found in linear time using an efficient technique as described in [2]. Then 4-point congruent sets are found by evaluating these point-pairs based on other invariances such as ratios and angles.

The hypotheses set returned by the above process by measuring the error between the returned pose hypotheses and the ground truth. For this purpose, a validation dataset containing 90 object poses was used. Specifically, in each hypothesis set, the pose hypothesis that has the minimum error in terms of rotation is selected as well as the one with the minimum translation error. The mean errors for these candidates over the dataset are shown in Table. 5.1. The results positively indicate the presence of hypotheses close to the true solution. Specifically, the candidate with the minimum rotation error seems almost perfect in the rotation and not very far even with respect to translation. Nevertheless, this hypothesis set contained approximately 20,000 elements. It is intractable to evaluate scene-level dependencies for that many hypotheses per object as the combined hypotheses set over multiple objects grows exponentially in size.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[All hypotheses] max. LCP score</td>
<td>11.16°</td>
<td>1.5cm</td>
</tr>
<tr>
<td>[All hypotheses] min. rotation error from ground truth</td>
<td>2.11°</td>
<td>2.2cm</td>
</tr>
<tr>
<td>[All hypotheses] min. translation error from ground truth</td>
<td>16.33°</td>
<td>0.4cm</td>
</tr>
<tr>
<td>[Clustered hypotheses] min. rotation error from ground truth</td>
<td>5.67°</td>
<td>2.5cm</td>
</tr>
<tr>
<td>[Clustered hypotheses] min. translation error from ground truth</td>
<td>20.95°</td>
<td>1.7cm</td>
</tr>
</tbody>
</table>

Table 5.1: Evaluating the quality of the hypotheses set returned by Super4CPS [2] with respect to different metrics.
5.3. Clustering of Hypotheses

To reduce the cardinality of the hypotheses sets returned by the subroutine SUPER4PCS in Algorithm 5, this work proposes to cluster the 6D poses in each set $T_O$, given a distance metric. Computing distances between object poses, which are defined in SE(3), in a computationally efficient manner is not trivial [109]. This challenge is further complicated if one would like to consider the symmetry of the geometric models, so that two different poses that result in the same occupied volume given the object’s symmetry would get a distance of zero.

To address this issue, a two-level hierarchical clustering approach is followed. The first level involves computing clusters of the pose set in the space of translations (i.e., the clustering occurs in $\mathbb{R}^3$ by using the Euclidean distance and ignoring the object orientations) using a K-Means process [110] to get a smaller set of cluster representatives $\text{cluster}_{tr}$. In the second level, the poses that are assigned to the same clusters are further clustered based on a distance computed in the SO(3) space that is specific to the object model, i.e., by considering only the orientation of the corresponding pose. The second clustering step uses a kernel K-Means approach [111], where the cluster representative is found by minimizing the sum of kernel distances to every other point in the cluster. This process can be computationally expensive but returns cluster centers that nicely represent the accuracy of the hypotheses set. By using this clustering method, the size of the hypotheses set can be reduced down from 20,000 rigid transforms in $T_O$ to 25 object pose hypotheses in $h_O$ for each object in the scene. The two bottom rows of Table 5.1 evaluate the quality of the cluster representatives in the hypotheses set. This evaluation indicates that the clustering process returns hypotheses as cluster representatives that are still close to the true solution. In this way, it provides an effective way of reducing the size of the hypotheses set without sacrificing its diversity.

5.4. Search

Once the hypotheses set is built for each object in the scene, the task reduces to finding the object poses that lie in the physically consistent neighborhood of the pose candidates and best explain the overall observed scene. In particular, given:
• the observed depth image $I_D$,
• the number of objects in the scene $N$,
• a set of 3D mesh models for these objects $M_{1:N}$,
• and the sets of 6D transformation hypotheses for the objects $h_{1:N}$ (output of Algorithm 5),

the problem is to search in the hypotheses sets for an $N$-tuple of poses $T_{1:N}$ so that $T_i \in f(h_i)$, i.e., one pose per object. The set $T_{1:N}$ should maximize a global score computed by comparing the observed depth image with the rendered image $R(T_{1:N})$ of object models placed at the corresponding poses $T_{1:N}$. Here, $f$ is the constrained local optimization of the object pose $h_i$ based on physical consistency with respect to the other objects in the scene and also the fact that the same points in the scene point cloud cannot be explained by multiple objects simultaneously. Then, the global optimization score is defined as:

$$C(I_D, T_{1:N}) = \sum_{p \in P} Sim(R(T_{1:N})[p], I_D[p])$$

where $p$ is a pixel (i,j) of a depth image, $R(T_{1:N})[p]$ is the depth of pixel $p$ in the rendered depth image, $I_D[p]$ is the depth of pixel $p$ in the observed depth image, $P = \{ p \mid R(T_{1:N})[p] \neq 0 \text{ or } I_D[p] \neq 0 \}$ and

$$Sim(R(T_{1:N})[p], I_D[p]) = \begin{cases} 1, & \text{if } |R(T)[p] - I_D[p]| < \epsilon \\ 0, & \text{otherwise} \end{cases}$$

for a predefined precision threshold $\epsilon$. Therefore, score $C$ counts the number of non-zero pixels $p$ that have a similar depth in the observed image $I_D$ and in the rendered image $R$ within an $\epsilon$ threshold. So, overall the objective is to find:

$$T^{*}_{1:N} = \arg \max_{T_{1:N} \in f(h_{1:N})} C(I_D, R(T_{1:N}))$$

At this point, a combinatorial optimization problem arises so as to identify $T^{*}_{1:N}$, which is
Algorithm 6: EXPAND($s_d,(M_{d+1},T_{d+1}),P_{d+1}$)

\begin{verbatim}
// $s_d$: state at depth $d$ (pose assignment for first $d$ objects)
// $(M_{d+1},T_{d+1})$: mesh model and pose hypothesis for the $(d+1)^{th}$ object
// $P_{d+1}$: point cloud segment for $(d+1)^{th}$ object
1 if $d = N$ then
  2 return NULL; // maximum depth of tree is reached
3 foreach $(M_O,T_O) \in s_d$ do
  4 $P_{d+1} \leftarrow P_{d+1} - POINTS_EXPLAINED(P_{d+1},M_O,T_O);$ // remove points from $P_{d+1}$ already assigned to an object $M_O$
  5 $T_{d+1} \leftarrow$TRIMMED_ICP(($M_{d+1},T_{d+1}),P_{d+1}$); // pose is locally refined using trimmed-ICP
  6 $T_{d+1} \leftarrow$PHYSICS_SIM(($M_{d+1},T_{d+1}),s_d$); // pose is locally refined based on physics simulation
  7 $s_{d+1} \leftarrow s_d \cup (M_{d+1},T_{d+1});$
  8 return $s_{d+1};$
\end{verbatim}

approached with a tree search process. A state in the search-tree corresponds to a subset of objects in the scene and their corresponding poses. The root state $s_0$ is a null assignment of poses. A state $s_d$ at depth $d$ is a placement of $d$ objects at specific poses selected from the hypotheses sets, i.e., $s_d = \{(M_i,T_i), i = 1:d\}$ where $T_i$ is the pose chosen for object $M_i$, which is assigned to a tree depth $i$. The goal of the tree search is to find a state at depth $N$, which contains a pose assignment for all objects in the scene and maximizes the above-mentioned rendering score. Alg. 6 describes the expansion of a state in the tree search process towards this objective.

The EXPAND routine takes as input the state $s_d$ at tree depth $d$, the point cloud segment corresponding to the next object to be placed, $P_{d+1}$, and the pose hypothesis $T_{d+1}$ for the next object to be placed $M_{d+1}$. Lines 3-4 of the algorithm iterate over the objects already placed in state $s_d$ and remove points explained by these object placements from the point cloud segment of the next object to be placed. This step helps in achieving much better segmentation, which is utilized by the local optimization step of Trimmed ICP (112) in line 5. The poses of objects in state $s_d$ physically constrain the pose of the new object to be placed. For this reason, a rigid body physics simulation is performed in line 6. The physics simulation is initialized by
Algorithm 7: SEARCH

// $M_{1:N}$: mesh models for all N objects
// $P_{1:N}$: point cloud segments for all N objects
// $h_{1:N}$: pose candidate sets for all N objects

Function MCTS ($M_{1:N}, P_{1:N}, h_{1:N}$)

1. $S \leftarrow \emptyset$
2. $L_{1:K} \leftarrow \text{GET_DEPENDENCY}(P_{1:N})$
   // $L_{1:K}$ is a partition of N objects into K subsets where objects belonging to different subsets are physically independent of each other
3. foreach $L \in L_{1:K}$ do
4.   $s_0 \leftarrow \emptyset$
5.   best_render_score $\leftarrow 0$
6.   best_state $\leftarrow s_0$
7.   while search_time $< t_h$ do
8.     // $t_h$ is a pre-defined time budget.
9.     $s_i \leftarrow \text{SELECT}(s_0, M_{1:N}, P_{1:N}, h_{1:N})$
10.    // $s_i$ is the next state to be expanded based on UCB
11.    \{s_N, R\} $\leftarrow \text{RANDOM_POLICY}(s_i, M_{1:N}, P_{1:N}, h_{1:N})$
12.    // $s_N$ is the state obtained by randomly selecting poses for all unplaced objects, i.e. not in $s_i$. R is the rendered score for $s_N$
13.    if $R > \text{best_render_score}$ then
14.      best_render_score $\leftarrow R$
15.      best_state $\leftarrow s_N$
16.    BACKUP_REWARD(s_i, R);
17.    // R is used to update estimated costs of all states $s$ along the path from $s_i$ to the root node.
18.   $S \leftarrow T \cup \text{best_state}$;
19. while
20. return S;
21. // $S$ is a set of object poses for N objects

inserting the new object into the scene at pose $T_{d+1}$, while the previously inserted objects in the current search branch are stationary in the poses $T_{1:d}$. A physics engine is used to ensure that the newly placed object attains a physically realistic configuration (stable and no penetration) with respect to other objects and the table under the effect of gravity. After a fixed number of simulation steps, the new pose $T_{d+1}$ of the object is appended to the previous state to get the successor state $s_{d+1}$.

The above primitive is used to search over the tree of possible object poses. The objective is to exploit the contextual ordering of object placements given information from physics and occlusion. This does not allow to define an additive rendering score over the search depth as
Algorithm 8: SEARCH MODULES

1 Function SELECT(s,M_{1:N},P_{1:N},h_{1:N})
   // s: state of the search tree
   // M_{1:N}: mesh models for all N objects
   // P_{1:N}: point cloud segments for all N objects
   // h_{1:N}: pose candidate sets for all N objects
   while depth(s) < N do
      if s has unexpanded child then
         d ← depth(s);
         T_{d+1} ← NEXT_POSE_HYPOTHESIS(h_d);
         // T_{d+1} is the next pose candidate for (d+1)th object that has not already been expanded for state s.
         return EXPAND(s,(M_{d+1},T_{d+1}),P_{d+1});
         // appends the pose T_{d+1} to state s
      else
         Return best child s according to UCB equation 5.1
      end if
   end while
   return s;

10 Function RANDOM_POLICY(s,M_{1:N},P_{1:N},h_{1:N})
   // s: state of the search tree
   // M_{1:N}: mesh models for all N objects
   // P_{1:N}: point cloud segments for all N objects
   // h_{1:N}: pose candidate sets for all N objects
   while depth(s) < N do
      d ← depth(s);
      T_{d+1} ← GET_RANDOM_HYPOTHESIS(h_{d+1});
      // T_{d+1} is a random pose assigned to the (d+1)th object.
      s ← EXPAND(s,(M_{d+1},T_{d+1}),P_{d+1});
      // appends the pose T_{d+1} to state s.
      return \{s, render(s)\};
   end while

16 Function BACKUP_REWARD(s,R)
   // s: state of the search tree
   // R: render score for the state s
   while s ≠ NULL do
      n(s) ← n(s) + 1;
      h(s) ← h(s) + R;
      // number of expansions and estimated cost of state s is updated
      s ← parent(s);
   end while

in previous work (71), which demands the object placement to not occlude any part of the already placed objects. Instead, this work proposes to use a heuristic search approach based on Monte Carlo Tree Search utilizing the Upper Confidence Bound formulation (108) to trade
off exploration and exploitation in the expansion process. The pseudocode for the search is presented in Alg. 7 and Alg. 8.

To effectively utilize the constrained expansion of states, an order of object placements needs to be considered. This information is encoded in a dependency graph, which is a directed acyclic graph that provides a partial ordering of object placements but also encodes the interdependency of objects. An example of a dependency graph structure is presented in Fig. 5.4. The vertices of the dependency graph correspond to the objects in the observed scene. Simple rules are established to compute this graph based on the detected segments $P_{1:N}$ for objects $O_{1:N}$.

- A directed edge connects object $O_i$ to object $O_j$ if the x-y projection of $P_i$ in the world frame intersects with the x-y projection of $P_j$ and the z-coordinate (negative gravity direction) of the centroid for $P_j$ is greater than that of $P_i$.

- A directed edge connects object $O_i$ to object $O_j$ if the detected bounding-box of $O_i$ intersects with that of $O_j$ and the z-coordinate of the centroid of $P_j$ in camera frame (normal to the camera) is greater than that of $P_i$.

The information regarding the independence of objects helps to significantly speed up the search as the independent objects are then evaluated in different search trees and prevent exponential growth of the tree. This results in a ordered list of objects, $L_{1:K}$ coming from the module GET_DEPENDENCY of Alg. 8, each of which is passed to an independent tree search process for pose computation. The MCTS proceeds by selecting the first unexpanded node starting from the root state. The selection of the next state to be expanded takes place based on a reward associated with each state. The reward is the mean of the rendering score received at any leaf node in the state’s subtree along with a penalty based on the number of times this subtree has been expanded relative to its parent. This is the Upper Confidence Bound (UCB) formulation ([108]). Formally, given a state $s$ of the search tree, the next state to be expanded is selected as,

$$s = \arg\max_{s' \in \text{succ}(s)} \left( \frac{h(s')}{n(s')} + \alpha \sqrt{\frac{2\log(n(s))}{n(s')}} \right)$$  \hspace{1cm} (5.1)
Figure 5.4: (Left) Image shows how a dependency graph is built based on interactions between the object segments. The Kleenex object is placed separately and does not need to be evaluated in the same tree like the others, whereas the placement of expo depends on the placement of crayola. (Right) The image shows one iteration of the Monte Carlo Tree Search process. The next state to expand is selected based on the previously computed score for the states. The selected state is then evaluated by executing a random policy which keeps expanding state until all objects in the current tree are placed. Finally, a rendering of this completely reconstructed scene is compared to the observed depth image to compute a score for the state.

where \( h(s) \) is the estimated score for state \( s \), \( n(s) \) is the number of times the subtree rooted at the state \( s \) has been expanded and \( \alpha \) is the parameter that controls the trade-off between exploration and exploitation in the search process. The selected state is then expanded by using a `RANDOM_POLICY`, which in this case is picking a random object pose hypothesis for each of the succeeding objects while performing the constrained local optimization at each step. The output of this policy is the final rendering score of the generated scene hypotheses. This reward is then back propagated in the step `BACKUP_REWARD` to all preceding nodes. Thus, the search is guided to the part of the tree, which gets a good rendering score but also explores other portions, which have not been expanded enough (controlled by the parameter \( \alpha \)). Figure 5.4 visualizes these steps of the MCTS pipeline.
5.5. Evaluation

This section discusses the dataset and metrics used for evaluating the approach, and an analysis of intermediate results explaining the choice of the system’s components.

5.5.1 Dataset and Evaluation Metric

A dataset of RGB-D images was collected and ground truth 6-DOF poses were labeled for each object in the image. The dataset contains table-top scenes with 11 objects from the 2016 Amazon Picking Challenge [113] with the objects representing different object geometries. Each scene contains multiple objects and the object placement is a mix of independent object placements, objects with physical dependencies such as one stacked on/or supporting the other object and occlusions. The dataset was collected using an Intel RealSense sensor mounted over a Motoman robotic manipulator. The manual labeling was achieved by aligning 3D CAD models to the point cloud extracted from the sensor. The captured scene expresses three different levels of interaction between objects, namely, independent object placement where an object is physically independent of the rest of objects, two-object dependencies where an object depends on another, and three object dependencies where an object depends on two other objects.

The evaluation is performed by computing the error in translation, which is the Euclidean distance of an object’s center compared to its ground truth center (in centimeters). The error in rotation is computed by first transforming the computed rotation to the frame attached to the object at ground truth (in degrees). The rotation error is the average of the roll, yaw and pitch angles of the transformation between the returned rotation and the ground truth one, while taking into account the object’s symmetries, which may allow multiple ground truth rotations. The results provide the mean of the errors of all the objects in the dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>No Dependency</th>
<th>2-Dependency</th>
<th>3-Dependency</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>APC-Vision-Toolbox</td>
<td>15.5°/3.4 cm</td>
<td>26.3°/5.5 cm</td>
<td>17.5°/5.0 cm</td>
<td>21.2°/4.8 cm</td>
</tr>
<tr>
<td>faster-RCNN+PCA+ICP</td>
<td>8.4°/1.3 cm</td>
<td>13.1°/2.0 cm</td>
<td>12.3°/1.8 cm</td>
<td>11.6°/1.7 cm</td>
</tr>
<tr>
<td>faster-RCNN+Super4PCS+ICP</td>
<td>2.4°/0.8 cm</td>
<td>14.8°/1.7 cm</td>
<td>12.1°/2.1 cm</td>
<td>10.5°/1.5 cm</td>
</tr>
<tr>
<td>PHYSIM-Heuristic</td>
<td>2.8°/1.1 cm</td>
<td>5.8°/1.4 cm</td>
<td>12.5°/3.1 cm</td>
<td>6.3°/1.7 cm</td>
</tr>
<tr>
<td>PHYSIM-MCTS (OURS)</td>
<td>2.3°/1.1 cm</td>
<td>5.8°/1.2 cm</td>
<td>5.0°/1.8 cm</td>
<td>4.6°/1.3 cm</td>
</tr>
</tbody>
</table>

Table 5.2: Comparing the accuracy of MCTS with different pose estimation techniques
5.5.2 Pose Estimation without Search

Evaluation was first performed over methods that do not perform any scene level or global reasoning. These approaches trust the segments returned by the object segmentation module and perform model matching followed by local refinement to compute object poses. The results of performing pose estimation over the collected dataset with some of these techniques are presented in Table 5.2. The APC-Vision-Toolbox [1] is the system developed by Team MIT-Princeton for Amazon Picking Challenge 2016. The system uses a Fully Convolutional Network (FCN) [51] to get pixel level segmentation of objects in the scene, then uses Principal Component Analysis (PCA) for pose initialization, followed by ICP [58] to get the final object pose. This system was designed for shelf and tote environments and often relies on multiple views of the scene. Thus, the high error in pose estimates could be attributed to the low recall percentage in retrieving object segment achieved by the semantic segmentation method, which in turn resulted in the segment not having enough information to compute a unique pose estimate. The second system tested uses a Faster-RCNN-based object detector trained with a setup-specific and autonomously generated dataset [6]. The point cloud segments extracted from the bounding box detections were used to perform pose estimation using two different approaches: i) PCA followed by ICP and ii) Super4PCS followed by ICP [58]. Even though the detector succeeded in providing a high recall object segment on most occasions, in the best case the mean rotation error using local approaches was still high (10.5°). This was sometimes due to bounding boxes containing parts of other object segments, or due to occlusions. Reasoning only at a local object-level does not resolve these issues.

5.5.3 Pose Estimation with the proposed approach

The proposed search framework was used to perform pose estimation on the dataset. In each scene, the dependency graph structure was used to get the order of object placement and initialize the independent search trees. Then, the object detection was performed using Faster-RCNN and Super4PCS was used to generate pose candidates, which were clustered to get 25 representatives per object. The search is performed over the combined set of object candidates and the output of the search is an anytime pose estimate based on the best rendering score. The
Figure 5.5: (Left) Rotation error in degrees and (Right) Translation error in cm as a function of the number of iterations.

The stopping criterion for the searches was defined by a maximum number of node expansions in the tree, set to 250, where each expansion corresponds to a physics simulation with Bullet and a rendering with OpenGL, with a mean expansion time of \(~0.2\) secs per node. The search was initially performed using a depth-first heuristic combined with the LCP score returned by the Super4PCS for the pose candidates. The results from this approach, PHYSIM-Heuristic (depth + LCP), are shown in Table 5.2, which indicates that it might be useful to use these heuristics if the tree depth is low (one and two object dependencies). As the number of object dependencies grow, however, one needs to perform more exploration. For three-object dependencies, when using 250 expansions, this heuristic search provided poor performance. The UCT Monte Carlo Tree Search was used to perform the search, with upper confidence bounds to trade off exploration and exploitation. The exploration parameter was set to a high value (\(\alpha = 5000\)), to allow the search to initially look into more branches while still preferring the ones which give a high rendering score. This helped in speeding up the search process significantly, and a much better solution could be reached within the same time. The plots in Fig. 5.5 capture the anytime results from the two heuristic search approaches.

### 5.5.4 Evaluating over Benchmark for Pose Estimation

The entire pipeline is evaluated over the Linemod ([36]) and the Linemod-Occluded ([39]) datasets. Evaluation is performed according to the benchmark ([3]) for 8 objects as
Figure 5.6: Example images from Extended Rutgers RGBD dataset and accompanying results from the Monte Carlo Tree Search process. The results are visualized in the light-weight physics engine (Bullet) which plays an integral part in performing the local optimization in this pipeline and ensures that the returned results are physically stable configurations.

shown in Fig. 5.7, which have corresponding ground-truth pose labels in both the datasets. The accuracy is measured in terms of the Visual Surface Discrepancy (VSD) metric as defined in [3] with a misalignment tolerance of $\tau = 20\text{mm}$ and correctness threshold $\theta = 0.3$. Given these parameters, the error is calculated by rendering the object model at the predicted and the ground-truth pose as depth maps $S$ and $S'$. These are compared to the actual depth map of the image to obtain visibility masks $V$ and $V'$ and the error is calculated as,

$$
e_{\text{vsd}} = \text{avg}_{p \in V \cap V'} \left\{ \begin{array}{ll} 0 & \text{if } p \in V \cap V' \land |S(p) - S'(p)| < \tau \\ 1, & \text{otherwise.} \end{array} \right.$$  

A pose is counted as correct if $e_{\text{vsd}} < \theta$. Finally, the recall rate per object and over the entire dataset are presented in Table 5.3 and Table 5.4.

To compare the proposed approach, first a synthetic training data was generated based on the developed pipeline. Some examples of the generated images are shown in Fig. 5.7. Overall 30,000 RGB and corresponding depth images are generated along with per pixel class labels. To generate this dataset, the intrinsic camera parameters, object texture and pose of the table are kept constant. The pose of the object over the table is varied randomly over x,y position and yaw while the rest of the pose parameters are kept constant. Finally, physics simulation
Figure 5.7: Performing pose estimation over Linemod and Linemod-Occluded dataset. The visualizations demonstrate (Left) the object models and final result of the pose estimation process on RGB-D data. (Middle) the training data generated from the proposed pipeline and (Right) some instances of successfull estimates as well as failure cases on Linemod-Occluded dataset with high level of occlusion.

is applied to get a physically-consistent scene which is rendered from 20 different viewpoints. The viewpoints are sampled randomly from a hemisphere of radius varying in a range similar to the test dataset. The camera sampling policy and range values are similar to the one used for generating the training data in [3]. Other scene parameters like light position, light color, object material emission, background and texture of the table are varied randomly within a pre-specified domain.

A Fully-convolutional network (FCN) ([51]) is trained with the generated data to obtain pixel-level classification and the output is used to guide the pose estimation process. The choice of using an FCN instead of Faster-RCNN was due to the fact that several unknown objects are present in the scene and predicting one definite location for an object in the scene would reduce the recall rate for the recognition task. In the Linemod dataset only one object needs to be estimated in each frame. To perform this task, first the pose of the table is computed using a RANSAC-based process and the direction of gravity is assumed to be perpendicular to the surface of the table. Then, 50 pose candidates are considered for the object based on the segmentation output and each of these are locally-optimized based on physics simulation and ICP in the MCTS process. Finally, the score is computed to select the best candidate. Due to the presence of unmodeled clutter, the optimization cost cannot assume that the entire scene can be explained by the estimated pose of known objects. Thus, the optimization cost for this dataset is set so as to maximize the alignment of the rendered depth map of the object at the predicted
<table>
<thead>
<tr>
<th></th>
<th>Ape</th>
<th>Can</th>
<th>Cat</th>
<th>Driller</th>
<th>Duck</th>
<th>Eggbox</th>
<th>Glue</th>
<th>Holepuncher</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hodan et al.</td>
<td>91</td>
<td>91</td>
<td>97</td>
<td>69</td>
<td>90</td>
<td>97</td>
<td>81</td>
<td>79</td>
<td>86.9</td>
</tr>
<tr>
<td>MCTS</td>
<td>93</td>
<td>90</td>
<td>87</td>
<td>90</td>
<td>80</td>
<td>97</td>
<td>80</td>
<td>65</td>
<td>85.3</td>
</tr>
<tr>
<td>Vidal et al.</td>
<td>89</td>
<td>92</td>
<td>96</td>
<td>89</td>
<td>87</td>
<td>97</td>
<td>59</td>
<td>69</td>
<td>84.8</td>
</tr>
<tr>
<td>Drost et. al.</td>
<td>86</td>
<td>93</td>
<td>87</td>
<td>92</td>
<td>66</td>
<td>96</td>
<td>53</td>
<td>67</td>
<td>80.0</td>
</tr>
<tr>
<td>Drost et. al.</td>
<td>77</td>
<td>98</td>
<td>94</td>
<td>96</td>
<td>45</td>
<td>94</td>
<td>68</td>
<td>66</td>
<td>79.8</td>
</tr>
<tr>
<td>Brachmann et al.</td>
<td>91</td>
<td>86</td>
<td>90</td>
<td>72</td>
<td>85</td>
<td>79</td>
<td>46</td>
<td>67</td>
<td>77.0</td>
</tr>
<tr>
<td>Hodan et al.</td>
<td>91</td>
<td>66</td>
<td>87</td>
<td>49</td>
<td>92</td>
<td>90</td>
<td>65</td>
<td>63</td>
<td>75.4</td>
</tr>
<tr>
<td>Brachmann et al.</td>
<td>74</td>
<td>88</td>
<td>66</td>
<td>81</td>
<td>69</td>
<td>66</td>
<td>50</td>
<td>75</td>
<td>71.1</td>
</tr>
<tr>
<td>Kehl et. al.</td>
<td>60</td>
<td>79</td>
<td>68</td>
<td>68</td>
<td>42</td>
<td>91</td>
<td>45</td>
<td>42</td>
<td>61.9</td>
</tr>
<tr>
<td>Buch et. al.</td>
<td>77</td>
<td>84</td>
<td>60</td>
<td>59</td>
<td>75</td>
<td>67</td>
<td>24</td>
<td>39</td>
<td>60.6</td>
</tr>
<tr>
<td>Buch et. al.</td>
<td>40</td>
<td>81</td>
<td>47</td>
<td>8</td>
<td>36</td>
<td>43</td>
<td>18</td>
<td>3</td>
<td>34.5</td>
</tr>
<tr>
<td>Buch et. al.</td>
<td>31</td>
<td>66</td>
<td>3</td>
<td>0</td>
<td>9</td>
<td>49</td>
<td>1</td>
<td>0</td>
<td>19.9</td>
</tr>
<tr>
<td>Tejani et. al.</td>
<td>36</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>70</td>
<td>27</td>
<td>0</td>
<td>18.3</td>
</tr>
<tr>
<td>Buch et. al.</td>
<td>11</td>
<td>3</td>
<td>7</td>
<td>18</td>
<td>12</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>8.1</td>
</tr>
<tr>
<td>Buch et. al.</td>
<td>3</td>
<td>9</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>4.0</td>
</tr>
<tr>
<td>Buch et. al.</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Table 5.3: Evaluating pose recall rate on the LINEMOD dataset according to the recent benchmark.

pose with the observed depth map. The alignment is computed with a distance threshold of 10mm and a surface normal tolerance of 30 degrees. The surface normal is used to avoid cases where the objects are falsely assigned to parts of large flat surfaces.

On the Linemod-Occluded dataset pose for 8 objects need to be estimated in every image with high level of occlusion. Two separate tests are performed on this dataset. In the first experiment the FCN is trained with just synthetic data from the proposed pipeline and the output is used to guide the MCTS process to estimate the pose for all 8 objects present in the scene. An example of the prediction is visualized in at the bottom-left of Fig. 5.7. In the second experiment, the FCN is re-trained with additional images from the Linemod dataset which are labeled using the confident estimates from the pose estimation over the entire dataset and projected to all the different views. Note that in this case only the segment corresponding to one object could be extracted from each image of the Linemod dataset, so a mask is used during the training process to only use that small part of the image which corresponds to the object and ignores the rest. This presents only positive samples for training on real-data and thus not a very significant improvement can be seen from this task. The performance corresponding to this experiment is referred to as the MCTS-SL in Table 5.4.

Overall, the proposed approach achieves state-of-the-art performance on both of these datasets. On the Linemod dataset, the proposed pipeline which is just trained on synthetic data
Table 5.4: Evaluating pose recall rate on the LINEMOD-Occluded dataset according to the recent benchmark

<table>
<thead>
<tr>
<th>Method</th>
<th>Ape</th>
<th>Can</th>
<th>Cat</th>
<th>Driller</th>
<th>Duck</th>
<th>Eggbox</th>
<th>Glue</th>
<th>Holepuncher</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL-MCTS</td>
<td>50</td>
<td>71</td>
<td>43</td>
<td>68</td>
<td>72</td>
<td>46</td>
<td>33</td>
<td>66</td>
<td>60.3</td>
</tr>
<tr>
<td>Vidal et al. [114]</td>
<td>66</td>
<td>81</td>
<td>46</td>
<td>65</td>
<td>73</td>
<td>43</td>
<td>26</td>
<td>64</td>
<td>59.3</td>
</tr>
<tr>
<td>MCTS</td>
<td>48</td>
<td>59</td>
<td>35</td>
<td>78</td>
<td>71</td>
<td>48</td>
<td>32</td>
<td>65</td>
<td>58.4</td>
</tr>
<tr>
<td>Drost et. al. [30]</td>
<td>62</td>
<td>75</td>
<td>39</td>
<td>70</td>
<td>57</td>
<td>46</td>
<td>26</td>
<td>57</td>
<td>55.4</td>
</tr>
<tr>
<td>Drost et. al. [30] (Edge)</td>
<td>47</td>
<td>82</td>
<td>46</td>
<td>75</td>
<td>42</td>
<td>44</td>
<td>36</td>
<td>57</td>
<td>55.0</td>
</tr>
<tr>
<td>Brachmann et al. [115]</td>
<td>64</td>
<td>65</td>
<td>44</td>
<td>68</td>
<td>71</td>
<td>3</td>
<td>32</td>
<td>61</td>
<td>52.0</td>
</tr>
<tr>
<td>Hodan et al. [47]</td>
<td>54</td>
<td>66</td>
<td>40</td>
<td>26</td>
<td>73</td>
<td>37</td>
<td>44</td>
<td>68</td>
<td>51.4</td>
</tr>
<tr>
<td>Brachmann et al. [49]</td>
<td>50</td>
<td>48</td>
<td>27</td>
<td>44</td>
<td>60</td>
<td>6</td>
<td>30</td>
<td>62</td>
<td>41.5</td>
</tr>
<tr>
<td>Buch et. al. [116] (ppfh)</td>
<td>59</td>
<td>63</td>
<td>18</td>
<td>35</td>
<td>60</td>
<td>17</td>
<td>5</td>
<td>30</td>
<td>37.0</td>
</tr>
<tr>
<td>Hodan et al. [37] (NR)</td>
<td>47</td>
<td>35</td>
<td>24</td>
<td>12</td>
<td>63</td>
<td>9</td>
<td>32</td>
<td>53</td>
<td>34.4</td>
</tr>
<tr>
<td>Kehl et. al. [44]</td>
<td>39</td>
<td>47</td>
<td>24</td>
<td>30</td>
<td>48</td>
<td>14</td>
<td>13</td>
<td>49</td>
<td>33.9</td>
</tr>
<tr>
<td>Buch et. al. [116] (si)</td>
<td>54</td>
<td>63</td>
<td>11</td>
<td>2</td>
<td>16</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>20.4</td>
</tr>
<tr>
<td>Buch et. al. [116] (ecsad)</td>
<td>29</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>9.6</td>
</tr>
<tr>
<td>Tejani et. al. [40]</td>
<td>26</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>4.5</td>
</tr>
<tr>
<td>Buch et. al. [117] (ppfh)</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2.3</td>
</tr>
<tr>
<td>Buch et. al. [117] (shot)</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.5</td>
</tr>
<tr>
<td>Buch et. al. [117] (ecsad)</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

achieves 85.3% accuracy that is just slightly below the template matching work of [37] (86.9%) in terms of overall success. Although template matching works well in cases of less occlusion, it fails to achieve a high recall on occluded datasets. Thus, on the Linemod-Occluded dataset, our proposed approach achieves the highest recall rate of 60.3% when the entire pipeline is used. When the self-learning component is not used, the performance is still just slightly below the top performing method of [114].

Some examples of the successful estimates and failure conditions on this dataset are presented in Fig. 5.7. One of the cases for failure is the presence of unmodeled objects on which the target object are physically dependent (Eggbox object in the bottom-right corner of the figure). The other failure case is that of object models getting good alignment scores with similar looking and large surfaces in the image (first three failure cases in the figure).

5.5.5 Limitations

One of the limitations of global reasoning as is in this approach is the large amount of time required for computing and searching over an extensive hypotheses set. Particularly, due to the hierarchical clustering approach, which was adapted to consider object specific distances, the hypotheses generation time for an object can be in the order of multiple seconds. The
search process, which seemed to converge to good solutions with 150 expansions for three-object dependencies, takes approximately 30 seconds. Nevertheless, both of these processes are highly parallelizable. Future work can perform the hypotheses generation and the search with parallel computing. Even though the use of bounding boxes to compute 3D point segments gives a high recall in terms of obtaining the object point segment, and the proposed system also addresses any imprecision which might occur in these segments by performing a constrained segmentation, sometimes an error which occurs in the initial part of object placement could lead to failures that need to be addressed.
Chapter 6

Scene-level Pose Estimation for Multiple Instances of Densely Packed Objects

The solution presented in this chapter [9] aims to improve the robustness of pose estimation for real-world applications, where object types appear multiple times, and in challenging, dense configurations, such as those illustrated in Figure 6.1, while allowing for scene-level reasoning. It aims to do so by proposing effective machine learning operations that depend less on manual data labeling and less on handcrafted combinations of multiple accuracy criteria into a single objective function. This allows the true automation of robot manipulation pipelines and brings the hope of wider-scale, real-world deployment.

Problem Setup: The considered framework receives as input: a) an RGB image \( x \) and a depth map of the scene; b) a set of mesh models \( \{M_i\}_{i=1}^K \), one for each object type \( i \in [1,K] \) present in the scene, and c) a set \( \{N_i\}_{i=1}^K \) expressing an upper-bound on the number of instances for each object type \( i \). The output is object poses as a set of rigid-body transformations \( \{\{T_{ij}\}_{j=1}^{N_i}\}_{i=1}^K \), where each \( T_{ij} = (t_{ij}^*, R_{ij}^*) \) captures the translation \( t_{ij}^* \in \mathbb{R}^3 \) and rotation \( R_{ij}^* \in SO(3) \) of the \( j^{th} \) instance of object type \( i \) in the camera’s reference frame, for each of the \( K \) object types present in the scene.

Figure 6.1 summarizes the considered pipeline: a) CNNs are used to detect semantic object classes and visible boundaries of individual instances, b) then, a large set of candidate 6D pose hypotheses are generated for each object class, c) quality scores are computed for each hypothesis, and d) scene-level reasoning identifies consistent poses that maximize the sum of individual scores. In the context of this pipeline, the contribution of this work relative to state-of-the-art methods is two-fold:

A. Adversarial training with synthetic data for robust object class and boundary prediction: Machine learning approaches have become popular in pose estimation, both in end-to-end
Figure 6.1: System pipeline and example output of the proposed approach on densely-packed scenes learning [48, 118] and as a pipeline component [39, 7]. They require, however, large amounts of labeled data. Recent approaches aim to solve single-instance pose estimation by training entirely in simulation [47, 7, 54]. The proposed method also utilizes labeled data generated exclusively in simulation to train a CNN for semantic segmentation. Nevertheless, CNNs are sensitive to the domain gap between synthetic and real data. The proposed training aims to mimic the physics of real-world test scenes and bridges the domain gap by using a generative adversarial network, as explained in Section 6.1. A key insight to improve robustness in the multi-instance case is that the network is simultaneously trained to predict object visibility boundaries. A thesis of this work is that boundaries learned on RGB images are more effective than boundaries detected on depth-maps [119] to guide and constrain the search for 6D poses, especially in tightly-packed setups.

B. Scene-level reasoning by automatically learning to evaluate pose candidate quality: This work finds the best physically-consistent set of poses among multiple candidates by formulating a constraint optimization problem and applying an ILP solver as shown in Section 6.4. The objective is to select pose hypotheses that maximize the sum of their individual scores, while respecting constraints, such as avoiding perceived collisions. Scene-level optimization has been previously approached as maximizing the weighted sum of various geometric features [68]. The weights characterizing the objective function, however, were carefully handcrafted. This work shows it is possible to learn the distance of a given candidate pose from a ground-truth one by using a set of various objective functions as features. A gradient boosted tree [120] is trained to automatically integrate these objectives and regress the distance to the closest ground truth pose. The objectives indicate how well a candidate hypothesis explain the predicted object segments, the predicted boundaries, the observed depth and local surface
normals in the input data. Prior related work has used tree search [107, 72, 8] to reconstruct the scene by sequentially placing objects. These prior approaches, however, were restricted to a small number of objects or fewer degrees-of-freedom due to computational overhead. The proposed \textit{ILP} solution is quite fast in practice and scales to a large number of objects. For the images of Figure 6.1, the scene-level optimization is achieved in a few milliseconds.

6.1. Semantic and Boundary Predictions

Fully Convolutional Networks (FCNs) [51, 121] are popular semantic segmentation tools. They have also been used for object contour detection [122] and predicting multiple instances of an object type [123, 124]. These networks are increasingly being trained in simulation [6, 53, 125, 47] to alleviate the need for large amounts of labeled data. The domain gap between the data generated in simulation and real data can lead to noisy predictions and greatly affect pose estimation accuracy. Several recent methods have been developed to bridge this domain gap [55, 56]. The current work subscribes to these ideas and: a) exploits the constraints available in robotic setups to simulate scenes with realistic poses, while b) uses adversarial training with unlabeled real images to bridge the gap between the labels predicted in synthetic data with those predicted in real ones.

We use a CNN to predict per-pixel semantic classification and a classification of whether a pixel is a visible object boundary. The data for training the CNN are generated in simulation with a physics engine and a renderer. The simulation samples a bin pose and a camera pose given the robot’s workspace. Each scene is created by randomly sampling, within a pre-specified domain,
the number of objects, the 6D pose of each object, the color of the bin, and the placement and intensity of the illumination sources. Finally, the scene is rendered to obtain a color image, a depth map, per-pixel class labels and visible instance boundary labels. The simulation generates a wide range of training data for domain randomization and robustness to domain gap issues. Nevertheless, domain gap still exists between synthetic data and data acquired through real sensors as it is hard to capture the full range of an object material’s interactions with various illumination sources in the environment.

The generative adversarial network (GAN), shown in Figure 6.2, performs the semantic and boundary detection tasks, while also adapting the output predictions on unlabeled real images to resemble the predictions on synthetic images. It consists of a shared VGG16 encoder that stacks five blocks of convolution, batch normalization and max pooling layers. The network branches out into two decoders, one for semantic classification and one for boundary classification. These are fully convolutional decoders with unpooling indices passed from corresponding max pooling blocks in the encoder section. The outputs of both decoders are passed to a corresponding fully convolutional discriminator network.

The network is trained by taking as input a synthetic image $x_s$ and its ground-truth label $Y(x_s)$. It also receives as input unlabeled real image $x_r$. The output $P_l(x_s)$ of the label prediction network on image $x_s$ (or corresponding output on $x_r$) is then passed on to the label discriminator $D_L$ whose task is to classify correctly if the prediction is on real or on synthetic data. Along with the objective of correctly labeling the synthetic image $x_s$, the label prediction network should also confuse $D_L$ into classifying $P_l(x_r)$ as an output of a sample coming from the synthetic domain. The objective of the semantic labeling network is defined as:

$$L_{sem} = L_{seg} + \lambda_a L_{domain}$$

$$L_{seg} = - \sum_{h,w} \sum_{k=1}^{K} Y(x_s)_{(h,w,k)} \log P_l(x_s)_{(h,w,k)}$$

$$L_{domain} = - \sum_{h,w} \log D_L(P_l(x_r))_{(h,w,0)}$$

where $P_l(x_s)$ is the per-pixel K-channel (for K object classes) output from the labeling network, $(h,w)$ are pixel coordinates, and $\lambda_a$ is a weight factor. $D_L(P_l(x))_{(h,w,0)}$ is the predicted score of
A similar GAN objective is used to simultaneously train the boundary predictor and discriminator.

### 6.2. Geometry-aware pose sampling

Given the output of semantic class predictions $P_l$ and boundary prediction $P_B$, this step aims to generate a set of 6D pose hypotheses $\{\{T_{ij}\}_{j=1}^{H_i}\}_{i=1}^{K}$, where $H_i$ denotes the number of pose hypotheses for each object model $M_i$ and is larger than the number of the object’s instances in the scene, i.e., $H_i \gg N_i$. Similar to the overall objective of the problem setup, each $T_{ij} = (t_{ij}, R_{ij})$ is a candidate translation $t_{ij} \in \mathbb{R}^3$ and rotation $R_{ij} \in SO(3)$ of some instance of object type $M_i$ in the camera’s reference frame, for each of the $K$ object types present in the scene.

The sets of pose hypotheses should be sufficiently large to ensure they contain candidates close enough to the unknown true poses, i.e., $\forall i \in [1,K], \forall j \in [1,N_i], \exists T \in \{T_{ij}\}_{j=1}^{H_i} : \|T - T_{ij}^g\| \leq \epsilon$, where $T$ is a hypothesis pose for object $M_i$ and $T_{ij}^g$ is the ground truth pose of the $j^{th}$ instance of object $M_i$. Since the ground truth poses are unknown during testing, one cannot guarantee that the previous desired property will be satisfied unless the candidates are densely sampled from $SE(3)$, which would make the search computationally expensive. Therefore, it is important to sample the candidates in a manner that balances their diversity and their similarity to the ground truth given the semantic class predictions $P_l$ and models $\{M_i\}_{i=1}^{K}$.

Pose hypotheses generation is often performed using RANSAC-like techniques [39] or via hough voting [30, 35]. These methods do not incorporate boundary information, which is important in cases where the surfaces of multiple object instances are aligned. The following explains the proposed pose hypotheses generation process (as illustrated in Figure 6.3), which utilizes both semantic segmentation and instance-boundaries.
Figure 6.3: A description of the stochastic optimization process for extracting the base $B = \{b_1, b_2, b_3, b_4\}$ so that it is distributed according to the stochastic segmentation and in accordance with the object’s known geometry. The base is matched against candidate sets $U = \{U_1, \ldots, U_N\}$ of 4 congruent points each from the object model $M$.

The output of the softmax layer from $P_i$ is used to compute $\pi_l(p_i)$, the probability of each pixel $p_i$ belonging to an object class $l$. This probability is defined as the ratio of the output $P_l(p_i)$ over the sum of the outputs for the same class over all pixels $p$ in the given test image $x$, i.e., $\pi_l(p_i) = \frac{P_l(p_i)}{\sum P_l(p)}$. For all pixels that are classified as a boundary pixel according to $P_B$, the probability $\pi_l$ is set to zero for all object categories. Additionally, for computational efficiency, $\pi_l$ is also set to zero for any pixel with a probability smaller than a threshold (set to 0.2 for all experiments). Given the corresponding depth image, each pixel is converted to a 3D point with its corresponding label probability. The point cloud is denoted by $S_l$.

Based on the principles of randomized alignment, candidate poses are generated for each instance of an object by sampling sets of points that belong to one instance with high probability. In the point cloud registration literature $[2, 106]$, a set of sampled points is called a base and denoted by $B$. Once sampled, this set is matched to congruent sets, denoted by $U$, on the corresponding object model $M$. Each of the congruent pairs $(B, U)$ defines a unique rigid transform as long as both $B$ and $U$ contain at least 3 points each. For computational reasons, the cardinality is typically limited to 4, i.e., $|B| = |U| = 4$. To maximize the probability that pose candidates for all the instances are generated, a number $A_i$ of base sets $\{B_{ij}\}_{j=1}^{A_i}$ on the image $x$ is sampled, for each object class $i \in \{1, \ldots, K\}$. Each base $B_{ij}$ can be matched to several congruent bases $U$ on the models, and thus defines a large set of candidate poses.

A. Sampling a Base from a Single Object Instance: The four 3D points that form a base $B = \{b_1, b_2, b_3, b_4 \mid b_{1,4} \in S_l\}$ are sampled from a point cloud $S_l$, which is the set of 3D points that are labeled as a category $l$ according to the semantic class probabilities $P_l$. The challenge here is...
to maximize the joint probability of the four points belonging to the same object instance. The joint probability distribution on the graph formed by the four points is represented by using the Hammersley-Clifford factorization and considering only unary and binary relations between the points:

$$Pr\left(B \rightarrow \text{Single instance of object class } l\right)$$

$$= \frac{1}{Z} \prod_{i=1}^{4} \left( \phi_{\text{node}}^l(b_i) \prod_{j<i}^{j} \phi_{\text{edge}}^l(b_i, b_j) \right),$$

where the unary terms $\phi_{\text{node}}^l(b_i)$ are the individual label probabilities of pixels $p(b_i)$ corresponding to 3D point $b_i$, i.e., $\phi_{\text{node}}^l(b_i) = \pi(p(b_i))$. The binary relations are defined as,

$$\phi_{\text{edge}}^l(b_i, b_j) = \begin{cases} 
1, & \text{if } \text{PPF}(b_i, b_j) \in \text{PPF}(M_k) \text{ and } \\
\text{shortest-path-length}(p(b_i), p(b_j)) < \varepsilon_l, & \text{otherwise.} 
\end{cases}$$

The Point-Pair Feature (PPF) for two base points $b_i, b_j$ with surface normals $n_i, n_j$ is defined as:

$$\text{PPF}(b_i, b_j) = (|| d ||, \angle(n_i, d), \angle(n_j, d), \angle(n_i, n_j)),$$

where $d = b_i - b_j$ is the vector from $b_i$ to $b_j$ and $n_i, n_j$ are local surface normals. PPF($M_k$) is the set of point-pair features pre-computed on the object model $M_k$ of object category $k$. The shortest-path-length term is the shortest distance on a graph $G$, where the vertices are the image pixels and an edge connects two pixels $p_i$ and $p_j$ if and only if: $P_B(p_i) < \delta$, $P_B(p_j) < \delta$, and $p_i$ and $p_j$ are adjacent pixels in the image (each pixel has at most eight adjacent pixels). In other terms, there is an edge between two adjacent pixels if both pixels have a low probability (less than a threshold $\delta$) of being on the boundary of an object. Thus, shortest-path-length($p_i, p_j$) is the length of the shortest path between the two nodes on $G$, and it is obtained using a breadth-first search.

**B. Avoiding Oversampling and Achieving Coverage:** The sampling process is dynamically adapted so as to avoid repetitively selecting the same image regions. In particular, after a base
is sampled, a decay factor $\gamma \in [0, 1)$ is multiplied to the potential $\phi_{\text{node}}^l(b_i)$ of every point $b_i$ that belongs to the same segment as $b_i$, i.e., $\phi_{\text{node}}^l(b_i) \leftarrow \gamma \phi_{\text{node}}^l(b_i)$. Points belonging to the same segments are those encountered during the breadth-first search. This encourages dispersion in the sampling process of the bases, so as to cover all instances of objects in the image and avoid having all the samples concentrated in the region of a single object instance. This becomes an issue when a particular, more prominent object instance has higher semantic class prediction probability than other instances.

C. Matching the Base to its Congruent Sets: Once all the base sets are sampled, a matching process is used for each one of the sampled bases $\{\{B_{ij}\}_{j=1}^{A_i}\}_{i=1}^{K}$ to compute a set $U_{ij} = \{u_1, u_2, u_3, u_4\}$ of 4-points from $M_i$ that are congruent to the sampled bases. The basic idea of 4-point congruent sets was derived from the fact that ratios and distances on objects are invariant across any rigid transformation [67].

This matching process generates a large number of pose candidates $\{\{T_{ij}\}_{j=1}^{H_i}\}_{i=1}^{K}$, several of which are redundant due to different congruent pairs returning similar transforms and the fact that most geometries have symmetry along multiple axes. Thus, this work follows a greedy approach to provide a diverse subset of the population. The approach sorts all pose candidates based on the Largest Common Pointset score [2] and picks pose representatives that are beyond a minimum distance from already selected poses. The distance here takes into account symmetry, and there are separate thresholds for rotation and translation, since these two variables operate in different spaces. The resulting set $\{\{T_{ij}\}_{j=1}^{H_i}\}_{i=1}^{K}$ of pose representatives then undergoes local refinement using ICP.

6.3. Pose Hypothesis Quality Evaluation

The objective of this step is to assign a single quality score $f(T)$ to each pose $T$ in the hypotheses set. The proposed score function considers five indicators $f_{1:5}(T)$ of a good alignment for each pose candidate $T$, shown in Table 6.1. The first three features $f_{1:3}(T)$, are indicative of the consistency of the hypothesis given a rendering of the object model placed at pose $T$. These are ratios in the range 0 to 1 that indicate the fraction of the visible boundaries or visible surface features of the rendering that are consistent with the observed depth image and boundary
Visible boundary for model hypothesis

\[ S_B \] Scene boundary pixels predicted by the CNN.

Visible portion of the object model when placed at pose T and rendered.

\[ V(T) \]

Label probability map.

\[ P \]

Distance between rendered and predicted boundary at pixel p.

\[ D_p \]

Depth distance between rendered and observed depth image at pixel p.

Scene boundary prediction

\[ \hat{S}_B \]

Observed depth map

Distance function over boundary prediction

Model-to-Scene consistency features

\[ f_1 = \frac{|B(T) \cap S_B|}{|B(T)|} \]: fraction of pixels in model boundary that match the scene boundary.

\[ f_2 = \frac{|B(T) \cap S_B|}{|V(T) \cap S_B|} \]: fraction of predicted boundary pixels in visible region that match model’s boundary.

\[ f_3 = \sum_{p \in V(T)} I_{D_S(p,T) < \delta_S} / |V(T)| \]: fraction of pixels in visible region of model placed at hypothesis that is sufficiently aligned in terms of depth to the observed data.

Scene alignment features

\[ f_4 = \sum_{p \in V(T)} P_l(p,T)S(p,T) \]: surface alignment score weighted by the corresponding label probability. Similarity score given by \( S(p,T) = (1 - \frac{1}{\delta_S}D_S(p,T))N(p,T) \) and it considers depth distance and surface normal similarity.

\[ f_5 = \sum_{p \in B(T)} (1 - \frac{1}{\delta_B}D_B(p,T)) \]: boundary alignment score based on distance between point on model boundary and it’s nearest point in the predicted boundary set.

Table 6.1: Description of features indicating good pose alignment with sensory input predictions. \( f_4(T) \) and \( f_5(T) \) correspond to the surface and boundary alignment scores based over respective distance functions. It is crucial to note that just maximizing over the alignment scores can lead to the selection of pose hypothesis that spans multiple instances in the scene. A straightforward solution to combine these features is to define \( f \) as a weighted sum of its components. Nevertheless, the resulting poses would heavily depend on the choice of the weights. Choosing the right weights manually for every new object type is not trivial.

Instead, a key aspect of this work is to learn the objective function \( f(T) \) using \( f_{1:5}(T) \) as features, i.e., \( f(T) = h(f_{1:5}(T)) \). The function \( h \) is learned by minimizing the following loss:

\[
\min h \sum_{(T,f_{1:5}(T),T^g) \in T_{train}} \left( h(f_{1:5}(T)) - e_{ADI}(T,T^g) \right)^2 \tag{6.1}
\]

where \( e_{ADI}(T,T^g) \) is the distance between a given pose \( T \) and a ground-truth pose \( T^g \). The distance \( e_{ADI}(T,T^g) \) is computed using the ADI metric, which is frequently used in the literature for evaluating pose estimations \cite{adi_metric}. This metric is explained in the empirical evaluation
The learn the objective function, a training data set is collected by simulating different scenes in a physics engine in the same way as described in Section 6.1. For each scene, the CNN predicts semantic and object boundaries and a large number of pose hypotheses are sampled. For each hypothesis \( T \), alignment features \( f_{1:5}(T) \) and corresponding scores \( e_{ADI}(T, T^g) \) are computed based on the closest ground-truth pose. The regression learning problem is to find a function \( h \) that maps features \( f_{1:5}(T) \) of a pose \( T \) to its actual distance from the corresponding ground-truth pose \( T^g \). In addition to the alignment features, the class of the object is given as a feature for the learning problem.

This work adopts the Gradient Boosted Regression Trees (GBRT) to solve the optimization problem in Equation 6.1. GBRTs are well-suited for handling variables that have heterogeneous features [120]. GBRTs are also a flexible non-parametric approach that can adapt to non-smooth changes in the regressed function using limited data, which often occurs when dealing with objects that have different shapes and sizes. An implementation of GBRT in the Scikit-learn library [126] is used in this work.

**6.4. Scene-level Pose Selection**

This section formulates the scene-level objective and presents how to address it in a computationally efficient manner, despite its computational hardness. The final objective is to select for an object category \( i \), a subset of poses \( \{T^{\star}_{ij}\}_{j=1}^{H_i} \subset \{T_{ij}\}_{j=1}^{H_i} \), which maximizes the total sum of scores \( f(T^{\star}_{ij}) \) while avoiding collisions between objects when assigned to those poses. In other terms, the intersection between the volume occupied by any two object instances in the scene should be empty. The approach first identifies a set that contains all pairs of poses that conflict with each other. This set is defined as \( \mathcal{C} = \{(i, j), (i', j')\} \) s.t. \( |\mathcal{V}(M_i, T_{ij}) \cap \mathcal{V}(M_{i'}, T_{ij'})| > \epsilon_v \) where \( i, i' \in \{1, \ldots, K\}, j \in \{1, \ldots, H_i\}, j' \in \{1, \ldots, H_{i'}\} \) and \( \mathcal{V}(M, T) \) is the volume occupied by a model \( M \) when placed according to pose \( T \). \( \epsilon_v \) is the maximum volume of tolerated collisions between objects in the scene. This positive error term is necessary because the best poses among the sampled ones may induce slight collisions between objects, which can often be corrected afterwards.
Pose selection is formulated as an integer linear programming problem, which can be solved by an ILP solver. A set of binary variables \( \{ y_{ij} \}_{j=1}^{H_i} \) is defined where \( y_{ij} \in \{0,1\} \). Each variable \( y_{ij} \) can be seen as an indicator variable on whether pose hypothesis \( T_{ij} \) is included in the set of selected posed \( \{ T_{ij}^{*} \}_{j=1}^{N_i} \). Then, the optimization problem is given as follows,

\[
\max_{\{y_{ij}\}} \sum_{i=1}^{K} \sum_{j=1}^{H_i} y_{ij} f(T_{ij})
\]

subject to:

\[
\forall i \in \{1,\ldots,K\}: \sum_{j=1}^{H_i} y_{ij} \leq N_i
\]

\[
\forall \{(i,j),(i',j')\} \in C: y_{ij} + y_{i'j'} \leq 1.
\]

The first constraint ensures that the number of poses selected for each category of object \( i \) does not exceed the number \( N_i \) of instances. The second constraint ensures that poses that are conflicting cannot be selected. This problem is equivalent to the Maximum-Weight Independent Set Problem (MWISP), with an additional constraint related to the number of selected poses. MWISP is NP-hard, and there are no \( \frac{1}{n^{1-\epsilon}} \)-approximations for any fixed \( \epsilon > 0 \) where \( n \) is the number of variables \( \{y_{ij}\} \). In practice, however, and for the problems considered here, an exact solution can be found very fast (in milliseconds) using modern ILP solvers for scenes containing a few hundreds of candidate poses. This is because the poses tend to cluster into cliques around specific instances, with a small number of constraints between poses in different cliques. Moreover, the objective function is monotone sub-modular as it is a linear function of a subset. Therefore, greedy optimization is guaranteed to find in linear time a solution that is at least \( (1 - \frac{1}{e}) \) fraction of the optimal. An approximate solution can also be found in linear time with an LP relaxation. After solving the ILP, the poses \( \{ T_{ij}^{*} \}_{j=1}^{N_i} \) are constructed from \( \{ T_{ij} \}_{j=1}^{H_i} \) by keeping those for which \( y_{ij} = 1 \).
6.5. Experiments

This section describes the experimental study performed over three datasets. Each dataset offers a different set of challenges for the pose estimation task. The first dataset is the bin-picking dataset \[129\]. It contains two object types and three scenarios (Figure 6.4). These scenarios correspond to clutter of similar looking objects with high occlusion rates. In this dataset, segmenting objects in color images is challenging due to their similarity in appearance, but depth cues can also be used to find an object’s boundaries. This leads to depth-based approaches achieving higher estimation success on this dataset.

To that end, a new dataset, henceforth called the densely-packed dataset is proposed in this paper. It comprises of 30 unique scenes with two object categories. Each scene contains 15 to 19 different instances of these objects, which were manually labeled with 6D pose annotations. There are two types of scenes in this dataset, as illustrated in Figure 6.4. In one type, denoted as scenario 1, the instances are tightly packed next to each other. This case is particularly challenging because the surfaces of multiple instances are aligned, which makes it difficult to use depth information for segmentation.

The third dataset used for the experiments is the occluded-linemod dataset \[39\]. It contains a sequence of 1214 frames from the linemod dataset \[36\], for which ground-truth pose labels are available for all 8 objects (of different categories) in the scene. The objects severely occlude each other in some views. The images also contain other objects for which no models are available. The presence of multiple classes and unmodeled objects makes it a suitable dataset to test the generality of the proposed approach.

In this study, the recall for pose estimation is measured based on the error given by the ADI metric \[36\]. It measures the distance between the predicted pose \(\hat{T}\) and the ground-truth pose
Given an object mesh model $M$:

$$e_{ADI}(\hat{T}, T) = \frac{1}{|M|} \sum_{x_1 \in M \cap \hat{x}_2 \in M} \min ||T(x_1) - \hat{T}(x_2)||_2,$$

where $T(x)$ corresponds to the transformation of a point $x$ that belongs to the object model.

Given the ground-truth pose, any prediction $\hat{T}$ is considered a true positive if $e_{ADI}(\hat{T}, T) < k_l d_l$. $k_l$ is a fraction set to $\frac{1}{10}$, while $d_l$ is the diameter of the object model calculated as the maximum distance between any two points on the model. The ADI metric accounts for symmetries in objects. This metric, however, only considers the geometry of the objects and ignores the alignment of the texture and color information. These attributes are however less important than geometry in robotic manipulation.

### 6.5.1 Implementation details

To train the proposed approach, synthetic images are generated via physics-based simulation and rendering with Blender-Python-API. The Cycles rendering engine in Blender is utilized to generate photorealistic images. The CNN in Section 6.1 is trained with 20,000 scenes generated via simulation. Adam optimizer with an initial learning rate of $2 \times 10^{-3}$ is used for the training. The weight $\lambda_a$ for the GAN loss is set to $10^{-3}$. To handle the class imbalance in the boundary network (as there are fewer boundary pixels than non-boundary pixels), the ratio of boundary to non-boundary pixels was computed in every iteration of the training and used to weight the respective loss terms.

In the pose hypotheses generation step, 100 base sets are sampled on the scene for each object category. The congruent set matching process returns a large number of pose candidates, which are then clustered with a similarity threshold of 3 cm and 30 degrees. The Gradient Boosting Regressor implementation from the Scikit-learn library is used for predicting the quality of individual pose hypothesis. The number of boosting stages and maximum depth of individual regression estimator are set to 1100 and 5 respectively. It is trained over 100 scenes generated by the same simulation process that was used for the trainig of the CNN. For each scene, approximately 1,000 pose candidates are generated for each object category, which results in 100,000 data points per category for training.
For the scene-level pose selection, the Gurobi optimization tool is used to solve the integer linear programming problem as described in previous section.

### 6.5.2 Experiments on bin-picking dataset

The bin-picking dataset [129] contains 183 RGB-D images with piles of *Coffee cups* and *Juice boxes* placed in a bin. There are three scenarios as shown in Figure 6.4. The task is to retrieve the pose for 15 instances of *Coffee cups* in the first scenario, 5 instances of *Juice boxes* in the second scenario, and 9 and 5 instances of the respective objects in the third scenario. The recall rate for the retrieval of object poses is reported in Table 6.2 based on the previously defined ADI metric.

<table>
<thead>
<tr>
<th>Approach</th>
<th>o1</th>
<th>o2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OURS</td>
<td>64.1</td>
<td>55.7</td>
</tr>
<tr>
<td>Buch et al. [116]</td>
<td>63.8</td>
<td>44.9</td>
</tr>
<tr>
<td>Drost et al. [30]</td>
<td>47.4</td>
<td>27.9</td>
</tr>
<tr>
<td>Doumanoglou et al. [129]</td>
<td>33.5</td>
<td>25.1</td>
</tr>
<tr>
<td>Tejani et al. [40]</td>
<td>31.4</td>
<td>24.8</td>
</tr>
</tbody>
</table>

Table 6.2: Pose estimation recall on the bin-picking dataset.

The results for competing approaches are obtained from [116]. As several object instances are completely hidden, the recall rates are lower, even for algorithms that could retrieve a large fraction of the visible instances. For the proposed solution, separate policies were used for data generation and separate networks were trained for each of the three scenarios. The 3D models of objects used in this dataset are created with surface reconstruction techniques such as *KinectFusion* [77] with VGA resolution images. The low quality of the reconstructed models, the existence of similar looking objects, and the randomness of the generated piles provide unfair advantages to techniques that are based only on depth information.

### 6.5.3 Experiments on densely-packed dataset

The densely-packed dataset introduces challenges for both RGB and depth-based techniques. Table 6.3 lists the recall rates for several popular pose estimation techniques on this dataset. The publicly shared implementations of these algorithms are run on the dataset to obtain these results. A synthetic version of the test dataset, with the same scenes as in the real dataset, was
generated to evaluate approaches that could not deal with the simulation-to-reality gap between
the training and the test data.

Template-matching \[36\] based on the templates extracted from color and depth rendering of
CAD models fails on several occasions as the templates are not robust to occlusion and varying
lighting conditions. Approaches based on Hough voting with point-pair features \[30, 116\]
depend only on depth-data. These approaches detect multiple object instances by considering
peaks in the Hough voting space, several of which are false positives due to aligned surfaces in
the packing scenario. Patch-based Hough forest technique \[129\] is tailored to handle multiple
instances of the same object category. It additionally performs joint registration and hypotheses
verification to reject false detections. Even after carefully tuning the weights of the optimization
function and relaxing the criteria for the number of pose candidates to select, the recall from
this approach is rather low.

<table>
<thead>
<tr>
<th>Approach</th>
<th>o1</th>
<th>o2</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing on real dataset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OURS</td>
<td>79.9</td>
<td>85.2</td>
<td>82.1</td>
</tr>
<tr>
<td>Hinterstoisser et al. [36]</td>
<td>37.0</td>
<td>65.6</td>
<td>49.3</td>
</tr>
<tr>
<td>Drost et al. [30]</td>
<td>30.1</td>
<td>57.6</td>
<td>41.9</td>
</tr>
<tr>
<td>Buch et al. [116]</td>
<td>11.2</td>
<td>31.7</td>
<td>19.9</td>
</tr>
<tr>
<td>Doumanoglou et al. [129]</td>
<td>16.2</td>
<td>44.3</td>
<td>28.3</td>
</tr>
</tbody>
</table>

| Testing on synthetic dataset |
| PoseCNN \[48\] | 15.0| 46.9| 28.7 |
| PoseCNN + ICP | 56.8| 80.6| 67.0 |
| DOPE \[47\] | 51.0| 70.6| 60.8 |

Table 6.3: Pose retrieval recall rate on densely-packed dataset.

**PoseCNN** \[48\] is an end-to-end learning-based approach. It includes a network branch for
semantic segmentation. Pixels belonging to an object class then vote for the object centroid’s
location. Based on the peaks in voting, the center is localized and corresponding inliers are
used to find a region of interest (RoI). Features within the RoI regress in a separate branch
of the network to output the object’s rotation. **PoseCNN** was originally developed for single
instances but via non-maximal suppression over the peaks of Hough voting, it can be adapted
for multiple instances. To eliminate the domain gap from the scope of testing, **PoseCNN** was
tested on the synthetic version of the test set. The output is used as an initialization for a depth-
based refinement process that utilizes perturbations and local search. Overall, object symmetry
Table 6.4: Pose recall on the occluded-linemod dataset. S: Synthetic data, R: Labeled real data, UR: Unlabeled real data.

and tight-packing scenarios make the simultaneous training of the multiple network branches hard to converge and the final refinement process less effective in these scenarios.

DOPE \cite{47} is another learning-based approach that recovers 6D poses via perspective-n-point (PnP) from predictions of 3D bounding-box vertices projected on the image. It aims to bridge the simulation-to-reality gap by a combination of domain randomization and photo-realistic rendering. The DOPE-DR version is considered for this evaluation due to its open-source availability. The neural network output is composed of a belief map used to find the projected vertices by a local peak search as well as an affinity map, which indicates the direction from projected vertices to their corresponding center for assignment. Nevertheless, when multiple instances of the same object are placed next to each other, some 2D vertices significantly overlap around the border of two neighboring instances. This makes the assignment of vertices to the correct center problematic and degrades the performance of PnP, since it requires relatively precise 2D-3D point correspondences. DOPE uses only color information without depth, which is a disadvantage when compared to other methods in Table 6.3. The network was trained from scratch with synthetic data generated by following the same pipeline as presented in \cite{47}. Using the best tuned parameters for domain randomization and post-processing steps, the pose estimation recall was measured on a synthetic validation set and on the real test set. The training fails to generalize to the real test set and achieves a very low recall rate of 3.5% and 10.5% on the two objects respectively.

For the proposed approach, synthetic training data is generated based on the known locations of the bin and the camera. Scenes are generated with two strategies 1) randomly dropping
the objects in the bin, 2) Lining up objects in the simulator according to placement poses generated based on their known dimensions to replicate packing scenario. Based on the training with this data and the proposed techniques for pose hypotheses generation and selection, a high recall rate could be achieved over the challenging test scenarios as indicated in Table 6.3. Two alternative strategies, reported in Table 6.5, were considered for estimating the object pose for multiple-instances as presented below.

1) **Mask-RCNN** $[52] + StoCS [7]**: One way to use StoCS for multi-instance pose estimation is to integrate it with an instance segmentation technique, such as Mask-RCNN. Mask-RCNN is trained with synthetic data and used to extract top 10 instances of each object type according to the detection probabilities. Pose estimation is performed using StoCS for each of these individual segments. There are two major limitations of this combination. The first is the use of deterministic instance boundaries leading to segmentation noise that cannot be recovered during pose estimation. The second is that visibility is not considered when matching the model to the detected segment, which can lead to several incorrect poses achieving high alignment scores.

2) **Sequential estimation**: This alternative considers sequentially selecting the object poses instead of the scene-level reasoning. It picks the most-likely pose based on the learned quality metric. The already selected pose is used as a constraint to select the next best hypothesis. This process continues until the maximum number of instances are selected for each object category or no hypothesis satisfies the constraint.

```
<table>
<thead>
<tr>
<th>Approach</th>
<th>o1</th>
<th>o2</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask-RCNN + StoCS [7]</td>
<td>42.8</td>
<td>68.3</td>
<td>53.7</td>
</tr>
<tr>
<td>Sequential estimation</td>
<td>68.3</td>
<td>83.6</td>
<td>74.9</td>
</tr>
<tr>
<td>OURS</td>
<td>79.9</td>
<td>85.2</td>
<td>82.1</td>
</tr>
</tbody>
</table>
```

Table 6.5: Alternative strategies considered during development.

```
<table>
<thead>
<tr>
<th>Method</th>
<th>Densely-packed dataset</th>
<th>Occluded-linemod dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>No adaptation</td>
<td>0.579</td>
<td>0.790</td>
</tr>
<tr>
<td>Cyclegan [55]</td>
<td>0.667</td>
<td>0.800</td>
</tr>
<tr>
<td>OURS</td>
<td>0.721</td>
<td>0.854</td>
</tr>
</tbody>
</table>
```

Table 6.6: Evaluating boundary prediction.
Figure 6.5: Evaluating semantic segmentation.

Figure 6.6: Examples of synthetic training data and boundary predictions on real images.

6.5.4 Experiments on occluded-linemod dataset

The occluded-linemod dataset contains 8 objects with different geometric shapes placed on a table-top and often severely occluded by one another. The challenge on this dataset is the presence of multiple object classes in the same image and the presence of objects in the background for which no labels are available. Table 6.4 compares the pose recall rate over each object with different RGB-D based techniques. The results for the competing approaches are reported based on [48]. Given that the objects have distinct geometry and are spaced over the table, depth based techniques [35] turn out to be effective on this dataset. Learning based techniques depend on labels over real images in addition to the geometric models to achieve similar performances.

The proposed approach can operate over unlabeled real images and achieve similar performance on the overall dataset. It outperforms all other approaches on 4 out of the 8 object categories, while it achieves relatively lower performance on objects such as glue and eggbox. To train for the occluded-linemod dataset, unlabeled real images from separate sequences of the linemod dataset were utilized. Other approaches use the same training data but with labels.

For the proposed solution, synthetic training images were generated by placing all 8 objects sequentially on the table-top. Translation along one axis and rotation around two axes are fixed,
such that the objects always stand on the table in simulation. The rest of the pose parameters are chosen at random within a domain, ensuring the objects remain on the table and not in collision with each other. The intrinsic camera parameters are known and the pose of the camera is sampled within a pre-defined domain as commonly done for this dataset. This ensures that the distribution of object poses in simulation is somewhat similar to what might exist in the real training data and helps bridge the simulation to reality gap.

### 6.5.5 Evaluating semantic and visibility boundary prediction

This section evaluates the accuracy of the semantic segmentation and the visibility boundary prediction from the CNN. The proposed network training strategy is first compared against training solely on synthetic data with no adaptation. Another alternative is an unpaired image-to-image translation network, **CycleGAN** [55]. Using this network, the synthetic images are first transformed to look similar to real images and then the labels available for the synthetic data are used to train the labeling network.

This evaluation is performed over two datasets, a multi-instance (*densely-packed dataset*) and a multi-class dataset (*occluded-linemod dataset*). It is observed that while CycleGAN is able to successfully transfer the appearance for objects from simulation to reality domain on the *densely-packed dataset*, it fails in the case of the *occluded-linemod dataset* due to the presence of multiple classes and background clutter. It was observed that the semantic information is lost during the image transformation process as the color and texture of some objects are changed.

Table 6.6 presents the average precision and recall for estimating the visibility boundary. The boundary prediction branch of the network returns a per-pixel classification (based on a

<table>
<thead>
<tr>
<th>Objective</th>
<th>Soap</th>
<th>Toothpaste</th>
<th>Ape</th>
<th>Can</th>
<th>Cat</th>
<th>Driller</th>
<th>Duck</th>
<th>Eggbox</th>
<th>Glue</th>
<th>Holepuncher</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>0.725</td>
<td>0.469</td>
<td>0.609</td>
<td>0.517</td>
<td>0.357</td>
<td>0.175</td>
<td>0.755</td>
<td>0.518</td>
<td>0.266</td>
<td>0.570</td>
</tr>
<tr>
<td>f2</td>
<td>0.776</td>
<td>0.750</td>
<td>0.770</td>
<td>0.633</td>
<td>0.607</td>
<td>0.675</td>
<td>0.835</td>
<td>0.647</td>
<td>0.454</td>
<td>0.860</td>
</tr>
<tr>
<td>f3</td>
<td>0.722</td>
<td>0.729</td>
<td>0.732</td>
<td>0.663</td>
<td>0.571</td>
<td>0.795</td>
<td>0.797</td>
<td>0.678</td>
<td>0.474</td>
<td>0.660</td>
</tr>
<tr>
<td>f4</td>
<td>0.583</td>
<td>0.852</td>
<td>0.796</td>
<td>0.844</td>
<td>0.627</td>
<td>0.905</td>
<td>0.835</td>
<td>0.568</td>
<td>0.564</td>
<td>0.875</td>
</tr>
<tr>
<td>f5</td>
<td>0.737</td>
<td>0.831</td>
<td>0.796</td>
<td>0.934</td>
<td>0.561</td>
<td>0.640</td>
<td>0.824</td>
<td>0.544</td>
<td>0.474</td>
<td>0.875</td>
</tr>
<tr>
<td>(f4 + f5)</td>
<td>0.598</td>
<td>0.826</td>
<td>0.839</td>
<td>0.949</td>
<td>0.596</td>
<td>0.835</td>
<td>0.851</td>
<td>0.673</td>
<td>0.590</td>
<td>0.910</td>
</tr>
<tr>
<td>Learned (f1:f5)</td>
<td>0.799</td>
<td>0.852</td>
<td>0.839</td>
<td>0.959</td>
<td>0.663</td>
<td>0.805</td>
<td>0.704</td>
<td>0.506</td>
<td>0.900</td>
<td></td>
</tr>
<tr>
<td>Optimal</td>
<td>0.876</td>
<td>0.908</td>
<td>0.887</td>
<td>0.974</td>
<td>0.841</td>
<td>0.99</td>
<td>0.782</td>
<td>0.694</td>
<td>0.975</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.7: Evaluating different objective functions for pose selection.
confidence threshold of 0.4) that indicates if a pixel is a visibility boundary pixel. The prediction is considered a true positive if it is within a threshold pixel distance (of 5 pixels) of any ground-truth boundary pixel. Figure 6.5 presents the precision-recall curve for pixel-wise semantic classification. The curve is plotted by considering different thresholds over confidence scores. The algorithm directly operates over the stochastic segmentation output for pose estimation. Figure 6.6 shows synthetic training images for different datasets and boundary predictions on real images of corresponding datasets.

6.5.6 Learned vs. Manually-defined objective functions

Table 6.7 compares the effect of using different objective functions for the scene-level selection of pose hypotheses. The evaluation is performed over the densely-packed and the occluded-linemod datasets. At first, each of the five individual features $f_1 : f_5$ are considered. $f_1 : f_3$ are fractions between 0 and 1 that measure the consistency of the pose candidates with respect to the semantic and boundary predictions. $f_4$ and $f_5$ are the alignment scores that respectively measure the surface alignment and the alignment of occlusion boundary between the pose hypothesis and the scene. A straightforward way to construct an objective function is to sum the two scene alignment scores ($f_3 + f_4$). It is not trivial, however, to combine the consistency scores with the alignment scores due to the difference in their scale and their significance. In general a weighted combination of these features could be used and manually tuned over different datasets. Alternatively, this objective function can be learned and as indicated in Table 6.7 it can be more effective than utilizing individual features. An upper bound is found for the pose selection process based on the entire hypotheses set. This is established by picking the hypotheses with minimum true ADI distance (as opposed to the predicted ones) from ground-truth poses.

Figure 6.7 shows the relative importance of the features in the construction of the boosted decision tree for predicting pose quality scores. The most important feature is the object’s class information for both datasets. The second most important feature turns out to be different in the two datasets. For the densely-packed dataset, it corresponds to the feature that computes the consistency of the model’s visible boundary to the boundary predictions. On the other hand, the surface alignment score is the second most important feature in the occluded-linemod dataset.
The results follow an intuitive explanation. The surface alignment scores are most significant in the occluded-linemod dataset as the objects have distinctive geometric features. This is not the case for densely-packed dataset where the surfaces of the objects are aligned and the occlusion boundaries are critical to select the best pose candidate.

Figure 6.8 shows the recall of the pose predicted using the learned objective function over varying distance thresholds. Given the ground-truth pose, a pose prediction is considered correct if the average distance ($e_{ADI}$) is less than 10% of the object’s diameter. This measure could be very stringent and more so for relatively smaller objects. The plot shows that a larger fraction of the objects can be retrieved given a relaxed distance threshold. A discrete set of pose hypotheses sometimes prevent the selection of the best individual candidates as they might be in collision. Another common failure is when a clutter object (not seen beforehand) is mis-detected as one of the target object. This occurs, for example in the occluded-linemod dataset for objects such as Glue and Eggbox as similar looking clutter objects are present in the scene and target objects are sometimes highly-occluded.
6.5.7 Ablation study

An ablation study is performed to demonstrate the importance of different components of the proposed solution. Given the predictions from the CNN, the hypotheses generation process finds for each object type a set of pose candidates that should be large enough to include the true poses and small enough to reduce the computation time. The number of candidates is affected by the number of sampled bases and the pose clustering process. Recall rates are shown in Figure 6.9 separately for all generated candidates and the ones selected by ILP. It also shows the effect of dispersion parameter $\gamma$, which is by default set to 0.9.

![Figure 6.9: Recall as a function of the number of pose candidates and the dispersion parameter $\gamma$.](image)

Table 6.8 evaluates the contribution of boundary reasoning in the hypothesis generation process. When the base sets are sampled without reasoning about the boundaries they often end up being sampled across multiple instances, leading to the generation of incorrect pose candidates. Only 78% of the instances have at least one pose candidate within acceptable distance. This further reduces the chances of their selection.

<table>
<thead>
<tr>
<th>Method</th>
<th>Selected</th>
<th>All-Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>with boundary</td>
<td>82.1</td>
<td>92.9</td>
</tr>
<tr>
<td>w/o boundary</td>
<td>58.7</td>
<td>78.0</td>
</tr>
</tbody>
</table>

Table 6.8: Effect of using boundaries for hypotheses generation.

Table 6.9 evaluates the contribution of the different types of data used in the training process.

In the first case, packed scenarios are not used as a part of the synthetic training data. Relatively lower performance indicates that the distribution of object poses in the training data is
crucial for the adaptation process. In the second scenario, no real data is used for the adaptation process. This implies that the CNN is trained solely on synthetic data with no adaptation. The stochastic nature of the hypotheses generation process and the robustness from scene-level selection could still recover a significant number of object instances. Nevertheless, the overall performance drops due to the presence of the domain gap.

6.5.8 Computation time

The mean and standard deviation of the computation time is evaluated over all scenes in the densely-packed dataset, and reported in Table 6.10. The overall computation time for the current sequential implementation of the approach ranges from 10s to 15s for estimating all (15 to 19) instances in a scene. The majority of the computation time is spent on depth-buffering and local refinement for each pose candidate (130 per object category). While the broad phase collision checking significantly helps to speed up the process, this step still remains a computational bottleneck compared to segmentation and ILP selection.

<table>
<thead>
<tr>
<th>Process</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmentation</td>
<td>0.005 ± 0.0004</td>
</tr>
<tr>
<td>Pose hypotheses generation</td>
<td>1.719 ± 0.4933</td>
</tr>
<tr>
<td>Clustering, refinement</td>
<td>1.995 ± 0.4567</td>
</tr>
<tr>
<td>Render and feature computation</td>
<td>10.147 ± 3.0022</td>
</tr>
<tr>
<td>Quality prediction</td>
<td>0.005 ± 0.0011</td>
</tr>
<tr>
<td>Collision checking</td>
<td>2.295 ± 0.7526</td>
</tr>
<tr>
<td>ILP selection</td>
<td>0.191 ± 0.0833</td>
</tr>
</tbody>
</table>

Table 6.10: Computation time for different components of the pipeline.

Several operations can be significantly sped up with parallel processing such as with the CUDA–OpenGL interoperability, which has been shown [72] to render 1000 images in 0.1s. Given the data parallelism, multiple cores can also be easily utilized to speed up the algorithm.
Chapter 7

Applications of Model-based 6D Pose Estimation

This chapter outlines a set of collaborative contributions towards applications of model-based object pose estimation. Section 7.1 focuses on bin packing application with a vision-based manipulation pipeline\cite{10}. Section 7.2 presents a simultaneous pose estimation\cite{12} for the adaptive hands used as end-effectors and for in-hand objects being manipulated. Section 7.3 designs an end-to-end learning based solution\cite{16} for tracking the pose of the object being manipulated by computing the residual poses between simultaneous frames. Section 7.4 involves a study\cite{11} of how object pose estimation can be performed when only weakly labeled images are available for training. Finally, Section 7.5 discusses a safe and effective strategy\cite{15} for picking objects in clutter when there is uncertainty regarding the pose of objects.

7.1. Robust Product Packing

The past decade has witnessed a vast growth of intelligent robotic solutions for logistics and warehouse automation tasks, with the Amazon Kiva mobile robotic fulfillment system serving as a prime example\cite{130}. Nevertheless, the completion of many tasks still rely on the use of repetitive human labor, such as for picking and packing of products and building customer orders. In particular, tightly packing products that are picked from an unstructured pile, the focal task of this work, still remains primarily manual, even though it is integral to warehouse automation and manufacturing.

Packing objects to fit in confined spaces, such as a small shipping box as the one in Fig. 7.1 is highly challenging. It can be argued that is more difficult than clearing clutter since packing requires placing objects in close vicinity to each other, in an ordered manner and also to be well aligned with the boundary of the container. This demands high levels of accuracy from the perception component as well as the manipulation strategy. Indeed, surprisingly little prior
work seems to exist that explicitly addresses this problem, let alone using a simple, suction-based end-effector.

To help narrow the application gap and enable the reliable, fully autonomous execution of such tasks, this system paper:

A. Proposes a complete hardware stack and an accompanying software pipeline for developing and testing algorithmic solutions for robot-enabled packing tasks. The hardware setup involves a single robotic arm as shown in Fig. 7.2, which depends only on depth-imaging technology to sense its environment. The result is a fully autonomous integrated system for picking objects from unstructured piles and placing them to satisfy a desirable packing arrangement, such as the one shown in Fig. 7.1.

B. Explores the use of a suction-based end-effector, which is also treated as a pushing finger, for product packing. The placement of objects, given the packing objective, requires the vacuum-based end-effector to pick objects from specific orientations, which may not be immediately accessible. Nevertheless, the paper demonstrates that the end-effector can still address such challenges in a reliable manner.

C. Develops and evaluates corrective prehensile and non-prehensile manipulation primitives that increase the pipeline’s robustness despite uncertainty regarding the pose of the objects. The uncertainty arises from the end-effector’s minimalistic nature and the use of only visual input. A critical aspect of the primitives is the intentional use of collisions, which exploit the inherent compliance of objects, the bins, and the end-effector for enhancing accuracy. Furthermore, the proposed primitives are tightly integrated with sensing to achieve in real-time:

i) toppling of objects in the initial unstructured bin to expose a desirable surface of the object for picking;

ii) pulling an object towards its target placement while pushing neighboring objects to further pack by operating directly over point cloud data; and

iii) real-time monitoring of potential failures and corrective pushing to achieve tight packing.

The evaluation uses the platform of Fig. 7.2. The experiments execute the pipeline in real-world scenes and show that the proposed manipulation primitives provide robustness despite the minimalism of the setup.
7.1.1 Problem Setup and Notation

Consider a robotic arm in a workspace $\mathcal{W}$ with known static obstacles, two bins $B_{\text{init}}$ and $B_{\text{goal}}$, as well as $n$ movable, uniform objects $\mathcal{O} = \{o^1, \ldots, o^n\}$ of known cubic dimensions. The bins $B_{\text{init}}$ and $B_{\text{goal}}$ are static but also compliant bodies in known poses that define a cuboid volume in $\mathcal{W}$ where the objects can be placed.

A labeled arrangement $A = \{p^1, \ldots, p^n\}$ is an assignment of poses $p^i \in SE(3)$ to each object $o^i$. Initially, the objects are in a start arrangement $A_{\text{init}}$, where the objects are inside $B_{\text{init}}$ in random but “stable” poses, i.e., the objects are stably resting and not moving. The $A_{\text{init}}$ arrangement is not known a-priori to the robot.

The objective is to move $\mathcal{O}$ to an unlabeled arrangement $\hat{A}_{\text{goal}} = \{\hat{p}^1, \ldots, \hat{p}^n\}$, which achieves a tight packing in $B_{\text{goal}}$. $\hat{A}_{\text{goal}}$ depends on the pose of $B_{\text{goal}}$, its dimensions and the object dimensions. The target arrangement and the picking order is input for the proposed process. The target arrangement maximizes the number of objects inside $B_{\text{goal}}$, while the objects rest on a stable face, and minimizes the convex hull of the volume the objects $\mathcal{O}$ occupy in $\mathcal{W}$. This is
Figure 7.3: *Left:* Pipeline in terms of control, data flow (green lines) and failure handling (red lines). The blocks identify the modules of the system. Sensing receives an RGBD image of $B_{\text{init}}$ and object CAD models to return a grasp point. Based on the picking surface, the object is either transferred to $B_{\text{goal}}$ or is handled by the Toppling module, which flips the object and places it back in $B_{\text{init}}$. When the object is transferred, a robust Placement module places the object at the target pose $\hat{p}$. The Packing module validates and corrects the placement to achieve tight packing. *Right:* a) Instance segmentation. b) Pose estimation and picking point selection are provided by sensing. c) Picking d) Toppling e) Placement and f) Packing.

typically a grid-like regular packing for cuboid objects. The unlabeled nature of $\hat{A}_{\text{goal}}$ means it is satisfied as long as one of the objects is placed at each target pose, i.e.,

$$\forall \hat{p} \in \hat{A}_{\text{goal}} : \exists \ p \in \mathcal{O} \ \text{so that} \ D(\hat{p}, p) < \varepsilon,$$

(7.1)

where $\varepsilon$ is a threshold for achieving the target poses; $D(\cdot, \cdot)$ is distance between object poses, which should consider the 3-axis symmetry of the cubic objects. In particular, if two poses result into an object occupying the same volume, then their distance is 0. For instance, rotation by $\pi$ about the vertical axis for a stably resting cube on a flat surface results in distance 0. A popular distance metric for 6D poses is the ADI metric $[36]$. The evaluation section will describe the distance used in the experimental process for evaluating the error of the final arrangement given point cloud data.

The arm has $d$ joints that define the arm’s configuration space $C_{\text{full}} \subset \mathbb{R}^d$, which has a collision-free subset $C_{\text{free}}$. Valid arm motions correspond to a continuous C-space curve $p : [0, 1] \rightarrow C_{\text{free}}$. The arm has an end-effector, such as a suction cup, for which discrete operations \{pick, release\} give rise to discrete modes: $\mathcal{M} = \{\text{transfer, transit}\}$. No within hand manipulation operations are available. The state space of the arm is: $\mathcal{X} = C_{\text{free}} \times \mathcal{M}$. Sensing is used to reason about the current object poses. Overall, the robot operations involve (i) moving the joints, (ii) picking or releasing objects and (iii) sensing.
The arm’s forward kinematics define a mapping $FK : C_{full} \rightarrow SE(3)$, which provides the pose $g \in SE(3)$ of the end-effector given $q \in C_{full}$. The reachable task space defines the end-effector poses the arm can reach without collisions: $T = \{q \in C_{free} : FK(q) \in SE(3)\}$. For the arm to pick $o^i$ at $p^i$, it has to be that the arm’s end-effector pose $g$ satisfies a binary output function: $\text{is\_pick\_feasible}(o^i, p^i, g)$, where $g \in T$. For instance, the pose $g$ of a suction cup must align it with at least one of the surfaces of an object $o^i$ at $p^i$. Then, it is possible to define the set of end-effector poses, which allow to pick an object at a specific pose:

$$G(o^i, p^i) = \{g \in T : \text{is\_pick\_feasible}(o^i, p^i, g) = \text{true}\}.$$

Assume the sets $G(o_i, p_i)$ are non-empty for all objects $O$ and poses in $A_{init}$ or $\hat{A}_{goal}$ (and their vicinity inside bins $B_{init}$ and $B_{goal}$). Otherwise, the task is not feasible. Note that it may be necessary to reconfigure the objects inside the initial bin so as to pick them from the appropriate face before placing them. This is due to the lack of within-hand manipulation.

Given the above definitions, the task is to identify a sequence of motions for the arm as well as end-effector and sensing operations to transfer the objects $O$ from the unknown initial stable arrangement $A_{init}$ inside $B_{init}$ to a tight, grid-based packing inside $B_{goal}$ defined by an unlabeled arrangement $\hat{A}_{goal}$, so as to satisfy Eq. 7.1.

### 7.1.2 System Components

This section describes critical components that influence the design of the proposed pipeline:

#### Hardware Setup

The robot used in the setup is the Kuka IIWA14 7-DoF arm (Fig. 7.2). A customized end-effector solution extrudes a cylindrical end-effector that ends with a compliant suction cup, to engage vacuum grasps. Two RealSense SR300 cameras are mounted on a frame and pointed to containers $B_{init}$ and $B_{goal}$ from the other side relative to the robot as Fig. 7.2 shows. While far from the robot, the cameras’ frame is statically attached to the robot’s base such that calibration errors are minimized in estimating the camera poses in the robot’s coordinate system.
**Workspace Design**

Fig. 7.2 shows the setup designed for the target task. The annular blue region represents the subset of the reachable workspace that allowed for top-down picks with the robot’s end-effector. This region is computed by extensively calling an inverse kinematics (IK) solver for top-down picks with the end-effector. The IK solutions indicate that the radial region between 40 cm and 70 cm from the robot center maximizes reachability and IK solution success given the setup. The bins (red rectangles) are placed so that they lie inside the optimal reachable region.

**Software**

MoveIt! [131] is used for motion planning. Most of the motions are performed using *Cartesian Control*, which guides the arm using end-effector waypoints. Ensuring the motions occur in reachable parts of the space increases the success of *Cartesian Control*, simplifies motion planning and speeds-up execution. To decrease planning time, motion between the bins is precomputed using the RRT* algorithm [132] and simply replayed at appropriate times.

### 7.1.3 Proposed Solution

The proposed pipeline, described in Fig. 7.3, provides the steps undertaken to perform the desired packing. The baseline steps correspond to:

a) *(Sensing)*: sense and select a target object $o^i$ to pick up;

b) *(Picking)*: execute action $\{\text{pick}\};$

c) *(Transfer)*: move $o^i$ to the next available target pose $\hat{p}^j$ and execute action $\{\text{release}\}.$

The experimental section considers this baseline pipeline, where it is observed that it performs poorly due to multiple sources of uncertainty, ranging from pose estimation and calibration errors, to object non-uniformity and object interaction with the bin and other objects, among others. These issue necessitate the introduction of remedies, which actively reduce the impact of the uncertainty. To this end, 3 manipulation primitives for a simplistic end-effector are designed to increase robustness and are integrated with the overall architecture:

i) Toppling;

ii) Robust Placement; and
iii) Corrective Packing.

7.1.4 Baseline: Pose Estimation and Picking

Given an RGB-D image of the source bin and a CAD model of the object, the objective is to retrieve \((o^i, p^i)\) such that it maximizes the chance of achieving target configuration \(\hat{p}^j\), where \(D(p^i, \hat{p}^j) \leq \epsilon\). To achieve this, the image is passed through a MaskRCNN convolutional neural network [52], which is trained to perform segmentation and retrieve the set of object instances \(O\). An image segment is ignored if it has a number of pixels below a threshold. It is also ignored, if MaskRCNN has small confidence that the segment corresponds to the target object. Among the remaining segments, instances \(o^i \in O\) are arranged in a descending order of the mean global Z-coordinate of all the RGBD pixels in the corresponding segment. Then, 6D pose estimation is performed for the selected instance [5][7].

If, given the detected 6D pose of the instance, the top-facing surface does not allow the placement of the object via a top-down pick, the next segment instance is evaluated in order of the mean global Z-coordinate. If no object reveals a top-facing surface, then the first object in terms of the maximum mean global Z-coordinate is chosen for picking.

For the selected object, a picking point, i.e., a point where the suction cup will be attached to the object, is computed over the set of points registered against the object model. It utilizes a picking-score associated in a pre-processing step with each model point, which indicates the stability of the pick on the object’s surface. The score calculates the distance to the center of the object mesh. A continuous neighborhood of planar pickable points is required to make proper contact between the suction cup and the object surface. Thus, a local search is performed around the best pick-score point to maximize the pickable surface.

7.1.5 Toppling

The toppling primitive is invoked if there exists no object that exposes the desirable top-facing surface, or if the object was erroneously picked from the wrong face. The latter is detected after the pick by performing pose estimation once the object is attached to the suction cup. For instance this can happen for the soaps shown in Figure 7.3(right)(d), if the thin side is available for pick but the soap needs to be placed on their wider side. In this case, a toppling primitive is
Figure 7.4: Adaptive pushing: (left) The black border is the bin and the gray rectangle a previously placed object. The green rectangle represents the target pose for the current object. The light green boundary represents an \( \epsilon \)-expanded model that intersects the point cloud at the purple points. These points result in the black vector that pushes the object away from them. (right) A screen shot of a scene’s point cloud, where the white points are collision points with previously placed objects and the red volume shows the computed pre-push pose for the new object.

Given a starting pose of an object \( p_{\text{start}} \) and a toppling action of the arm, the object ends up at a new pose \( p_{\text{topple}} \). The objective is to allow the existence of a pick \( \hat{g} \in G(o^i, p_{\text{topple}}) \) so that there is a transfer action from pick \( \hat{g} \) that achieves the final placement \( p_{\text{end}} \) close enough to the desired target placement \( \hat{p}^j \), i.e., \( D(p_{\text{end}}, \hat{p}^j) \leq \epsilon \). For the considered setup, this means that the top-facing surface of the object at \( p_{\text{topple}} \) and \( \hat{p}^j \) is the same.

The toppling module inspects the visible point cloud in the source bin to identify the best toppling plane, which is sufficiently empty and flat. The accompanying implementation restricts actions to ones that change the pose of the cubic objects by shifting the most upward facing surface to only one of the adjacent surfaces. While this does not guarantee toppling the object to all possible poses, the symmetry of cubic products resolves this issue.

Prior work \cite{133} has shown the efficacy of minimal end-effectors used in tandem with the environment to achieve toppling. In the previous work, the friction against a conveyor belt is used to topple an object about a resting surface. The conveyor belt’s motion is parallel to the initial resting surface plane. In the current setup, the compliance of the suction cup is used to emulate the same effect using a lateral motion on the same plane as the top-surface along the direction of the desired transformation between \( p_{\text{start}} \) and \( p_{\text{topple}} \). Due to symmetry, at least one neighboring surface allows top-down picks, so a successful toppling action exists. Using the pose of the object, the lateral motion direction is executed once the object is placed on the
detected plane, and the object is released during this action. Results show that this is highly effective in the target setup.

### 7.1.6 Point Cloud Driven Adaptive Pushing

Directly placing objects at the goal pose $\hat{p}^i$ into bin $B_{\text{goal}}$ is prone to placement failures due to errors in perception as well as prior placements. This may result in damaging the objects. A safer alternative is to drop the object from a certain height, right above the goal pose, so as to avoid pressing against previously erroneously placed objects. Still, however, this alternative results in low quality packing. A key realization is that during placement, the object being manipulated will inevitably approach or even collide with other objects or the target container. To sidestep undesirable collisions, an adaptive pushing primitive is developed, which directly operates over point cloud data for the target bin.

The process is shown in Fig. 7.4. The adaptive pushing begins by growing the object model at the target pose $\hat{p}^i$. Given the uncertainty value $\varepsilon$, the model is enlarged by $2\varepsilon$ along each dimension. The enlarged model is intersected with the point cloud to retrieve collision points. A collision vector is computed as a vector pointing from a collision point to the center of the object model. Summing all collision vectors yields the displacement vector. By iteratively moving the object model along the unit displacement vector, where the displacement vector is recomputed after each movement, a collision-free pre-push pose for the object is obtained. During the execution, the robot first moves the object above the pre-push pose, then lowers the object to the pre-push pose and finally pushes the object to the target pose.

### 7.1.7 Fine Correction using Push and Pull Primitives

The final primitive deals with the remaining failure cases. Fine corrections are required because objects can be placed in incorrect poses due to unexpected collisions as well as calibration and pose estimation errors. The proposed corrective manipulation procedure continuously monitors the scene and triggers corrective actions whenever necessary.

The process first removes the background, the box, the robot’s arm and end-effector from
the observed point cloud and computes its surface normals. The observed point cloud is then compared against the desired alignment of the objects in their target poses. As shown in Fig. 7.5, two types of misalignment errors can occur. The first type occurs when the top surface normal of an object is not perpendicular to the support surface. This error is corrected by pushing the object along a direction and for a distance computed based on its surface normal in a manner that makes it aligned with the support surface. The second type of error happens when a peripheral object is not entirely within the desired footprint of the pile. The proposed procedure systematically detects pivot points that are outside the desired footprint of the pile and pulls their objects inside accordingly. The correction is repeated until the point cloud is aligned with the desired goal poses given a threshold $\epsilon$, or a timeout occurs.

7.1.8 Evaluation

To evaluate the performance of the proposed pipeline, extensive experiments were executed on the Kuka platform of Fig. 7.2 to accurately represent the motivating complications that arise in real-world setups. The experiments are designed to showcase the hardness of tight packing as well as the benefits of adding robust environment-aware manipulation primitives that aid in increasing success rate and accuracy.

For consistency, an identical version of the problem is tested, with “dove soap bars” that are randomly thrown into the source bin $B_{\text{init}}$, which is placed on one side of the robot’s reachable workspace (Fig. 7.2). Only top-down grasps are allowed within a given alignment threshold. The start arrangement $A_{\text{init}}$ of objects is intended to reflect a random pile, with 10 repetitions of each experimental condition. The target bin $B_{\text{goal}}$ contains a $3 \times 3$ grid arrangement of 9 objects, on the same plane, with the stable face of the object targeted for placement. The complete pipeline uses a) corrective actions for fine adjustments, b) push-to-place actions for robust placement, c) toppling actions for increasing successes, and d) pose estimation for adjusting the object. The improvements introduced by these strategies are evaluated through the following comparison points, within the context of the proposed pipeline:

**V1 - Full pipeline**: The complete pipeline with all the primitives achieves the highest accuracy and success rate.

**V2 - No corrective actions**: The experiment corresponds to the use of V1 without the fine
Figure 7.6: (left) The final set of object poses in the target poses at the end of every experiment. Different column represents different versions. The top row is the best case, and the bottom row is the worst case. (middle) the blue bar represents the fraction of successful object transfers, the orange bar represents the percentage of unoccupied volume within the ideal target placement volume. (right) the blue bar represents the average number of correction actions happened per experiment, the green bar represents the average number of toppling actions happened per experiment.

V3 - No push-to-place actions: This version is V2 without the use of the robust placement module (Fig. 7.4) that performs push actions to achieve robust placement.

V4 - No toppling actions: These experiments used V2 without considering toppling actions to deal with objects not exposing a valid top surface that allows the target placement.

V5 - (Baseline) No push-to-place, toppling, pose-estimation: The naive baseline that solely uses a pose-unaware grasping module that reports locally graspable points and drops the grasped object at an end-effector pose raised from the center of the desired object position, with no adjustment in orientation.

The metrics evaluated include the fraction of successful object transfers that succeed in moving objects to the target bin. The accuracy is captured in the threshold mentioned in Eq. (7.1) that is expressed in terms of a percentage of unoccupied volume within the ideal target placement volume. This was measured with a voxel discretization sufficient to elucidate the difference between the methods. The average data recorded is reported in Fig. 7.6. This error measure is proportional to the accuracy. The points to note for every version are detailed as follows.

The low error for V1 corroborates the final bin placement evidence. On average 7 corrective actions per experiment were invoked to achieve the high degree of accuracy.

The accuracy improvement obtained from corrective actions is evaluated in V2. While this
version succeeded in dropping all the objects close to the correct target poses, application use-cases where a higher degree of accuracy is desired motivate the use of corrective actions. The integration of the corrective actions was done with higher error threshold during intermediate steps, and a much finer one for the final adjustment. Errors can typically arise from execution failure and pose misalignments. The less accurate these underlying processes are, the more important corrective actions become.

**V3** only performs adjustments using pose estimation, and toppling. While, this is sufficient to successfully transfer all the objects, any difference of accuracy to **V2** would be introduced by the lack of push-to-place actions. Here there is complete reliance on the exactness of the execution and pose adjustments. Due to the proximity of adjacent object surfaces in the target grid arrangement, even minor errors get aggravated. However, due to the ability to reason about toppling, all the objects can be transferred to the target bin, even with this low accuracy. This is demonstrated in the occurrence of the failure to transfer all the objects.

In **V4** any object that does not expose an permitted picking surface that makes the prehensile placement possible, is not picked. Any instance of the source bin, which ends with no such objects results in no valid picking actions that can make the approach proceed, and a failure is declared. The current behavior of **V4** drops the object if it is mistakenly grasped from the wrong surface. This can itself be used as a naive toppling primitive. It is important to note that there might be other alternative strategies that can deal with this failure, but the intent of this comparison is to demonstrate the importance of having a deliberate toppling strategy in the pipeline, that can change the object’s orientation in the context of random starting arrangements of the object. On average, over **V1, V2, and V3** the toppling primitive was required 4 times per experiment. This highlights the necessity of this reasoning. Deliberate toppling however requires at least one additional pick action. The number of pick attempts per successful object transfer was 2.56 for **V4**, whereas, in **V2** the same was 1.98. This indicates that toppling is indeed necessary both in terms of success and efficiency of actions.

Expectedly, **V5** has the lowest accuracy. However, since there is no reasoning about the pick surface, every object was transferred to the space of the bin. This has no guarantee to work if the object is larger. This drives the motivation for using a set of robust primitives for the packing problem.
Overall, the time for the experiments show a trend of increasing with the increasing complexity of the pipeline. The trade-off of accuracy versus time persists. On average, V1 ran for 945s while V5 ran for 323s.

7.2. In-hand Pose estimation

Many manipulation tasks, such as placement or within-hand manipulation, require the object’s pose relative to a robot hand. The task is difficult when the hand significantly occludes the object. It is especially hard for adaptive hands (Fig. 7.7), for which it is not easy to detect the finger’s configuration. In addition, RGB-only approaches face issues with texture-less objects or when the hand and the object look similar. This paper presents a depth-based framework, which aims for robust pose estimation and short response times. The approach detects the adaptive hand’s state via efficient parallel search given the highest overlap between the hand’s model and the point cloud. The hand’s point cloud is pruned and robust global registration is performed to generate object pose hypotheses, which are clustered. False hypotheses are pruned via physical reasoning. The remaining poses’ quality is evaluated given agreement with observed data. Extensive evaluation on synthetic and real data demonstrates the accuracy and computational efficiency of the framework when applied on challenging, highly-occluded scenarios for different object types. An ablation study identifies how the framework’s components help in performance.

7.2.1 Problem Formulation

Given a depth image from camera C, a mesh model M of object O, the goal is to compute O’s 6D pose, i.e., the rigid transform $T_{CM}^C$, where O is grasped by an adaptive hand in C’s view.
Figure 7.8: The framework acquires the RGB-D point cloud and computes the configuration of the adaptive hand given its CAD model. From this estimate, the hand is removed from the point cloud and the object segment is recovered. A set of pose candidates is generated by matching the segment to the object’s model. The most likely pose is returned by evaluating the consistency of the interactions between the estimated hand and the in-hand object.

The work considers under-actuated hands (the Yale Hand T42 [134]) for which a CAD model is available. The hand state determined by configuration of the $N$ fingers $x_H = \{q_F\}_{i=1}^N$ are initially unknown and not available. The camera is calibrated and the transform $T_{HC}$ of the hand’s wrist frame $H$ to the camera is available.

Fig. 7.8 outlines the proposed approach with 3 key components: 1) parallel evolutionary optimization to estimate the hand’s configuration; 2) heuristics-guided global pointset registration to generate pose hypotheses for the object; 3) scene-level physics reasoning that considers the hand-object interaction to find the most-likely object pose.

### 7.2.2 Hand State Estimation

An adaptive hand consists of a wrist and a set of fingers. The fingers are not sensorized to a level that provides reliable state information. Each finger $F$ is treated as an articulated chain and its configuration is the set of all joint angles, i.e., $q_F = \{\theta_{F1}, \theta_{F2}, ..., \theta_{FR}\}$ (see Fig. 7.9). A 3D region-of-interest (ROI) is identified that contains the point cloud $P_S$ of the in-hand object and fingers. The ROI is computed based on the wrist’s pose $T_{HC}$ obtained from forward kinematics and the hand dimensions. ICP, performed over the point cloud and the wrist’s model, refines $T_{HC}$ to compensate for errors in forward kinematics and camera calibration.

The next step aims to find the finger configuration, which minimizes the discrepancy between the robot hand model and the observed depth image given $P_S$. It is possible to formalize
this problem as convex objective optimization and employ gradient descent algorithms to obtain the optimal pose, as in related work [135]. An initial estimation from the previous frame, in the context of a tracking scenario, can be good initialization for the gradient descent to converge. Nevertheless, in single image estimation, such as in this work, no such initial guess is assumed. For this reason, this paper proposes Particle Swarm Optimization (PSO) for searching each finger configuration, inspired by prior work on human hand pose tracking [136]. PSO is an evolutionary process where particles interact with each other to search the parameter space. In addition to being less sensitive to local optima, it is highly parallelizable and does not require the objective function to be differentiable. This allows to formalize the cost function as minimizing the negative LCP (Largest Common Pointset) [67] score computed via an efficient KDTree implementation.

Unlike human hands, the configuration space of robot hands is more constrained. It was empirically observed that instead of estimating the hand state globally in PSO, sequentially estimating each finger’s configuration leads to more stable solutions and faster convergence (with 15 particles and 3 iterations for each finger). Therefore, PSO was applied to each finger separately to estimate its configuration starting from the finger closest to camera. Each PSO particle is a vector representing the current finger configuration $q_F$ and the swarm is a collection of particles. Initially, particles are randomly sampled and their velocities are initialized to zero. In each generation, a particle’s velocity is updated as a randomly weighted summation of its previous velocity, the velocity towards its own best known position, and the velocity towards the best known position of the entire swarm.

The cost function evaluation is given in Alg. The inputs are the finger configuration $q_F$, which will be evaluated, the hand region point cloud $P_S$ and finger model point cloud $P_F$. In lines 2 - 5, a penalty is assigned to cases when fingers have collisions. It returns a score that is linearly dependent on the penetration depth $d$ to encourage particles to move to a more promising parameter space that satisfies collision avoidance. The $\lambda_c$ parameter is a penalization term and is arbitrarily assigned to a very large value. $P_S$ is first transformed into the finger frame using forward kinematics and $q_F$. A KDTree is built on the transformed $P_S$ to compute the LCP score with the finger model cloud efficiently.

The single shot hand state estimation is implemented for parallel execution in C++. This
Algorithm 9: COST_FUNCTION \((q_F, P_S, P_F)\)

1. \(P_{S}^{\text{finger}} \leftarrow \text{transform } P_S \text{ to finger frame using forward kinematics and } q_F;\)
2. for any other finger \(Q_F\) do
   3. \(d \leftarrow \text{collisionCheck}(P_F, Q_F);\)
   4. if \(d < \varepsilon\) then
      5. return \(-\lambda_c - \lambda_d;\)
   6. \(\text{kdtree}(P_S) \leftarrow \text{build kdtree from } P_{S}^{\text{finger}};\)
   7. \(LCP \leftarrow 0;\)
3. for each \(p_F \in P_F\) do
   4. \(p_{\text{nei}} \leftarrow \text{kdtree}(P_S).\text{findNearestNeighbor}(p_F);\)
   5. if \(\|p_{\text{nei}} - p_F\| < \varepsilon \text{ and normal}(p_{\text{nei}}) \cdot \text{normal}(p_F) > \delta\) then
      6. \(LCP \leftarrow LCP + 1;\)
   7. return \(-LCP;\)

component can also be very useful for tracking approaches \([135, 137]\) as initialization or re-initialization.

### 7.2.3 Object Pose Hypotheses Generation and Clustering

Once the full hand state \(x_H\) is available, \(SDF\) (Signed Distance Field) is computed for the hand. All \(P_S\) points with signed distance below a threshold \(SDF(p, x_H) < \varepsilon\) are eliminated \((\varepsilon = 3 \text{ mm} \text{ in the accompanying experiments})\). The remaining point cloud \(P_O\) is now assigned to the object. The new goal is to register the object mesh \(M_O\) against the point cloud \(P_O\), despite the imperfections of \(P_O\) due to sensor noise, occlusions or errors in the hand state estimate.

This paper builds upon prior work for hypotheses generation \([2, 7]\). It samples sets of 4-point, co-planar bases on the object’s point cloud \((P_O)\), and searches for congruent sets on the object model \((M_O)\) to provide a pool of rigid alignments (Fig. 7.10). Bases can be sampled randomly \([2]\) or given the stochastic output of a CNN \([7]\). To limit the number of samples, while maximizing the chances of sampling a valid base (where all points belong to the object), this work proposes sampling heuristics given the hand state.

The base sampling process is given in Alg. 10, where inputs are the object point cloud \(P_O\), heuristics \(\pi\) and a hash map \(PPF_M\) of Point Pair Features (PPF) \([30]\) of the model \(M_O\). The hash map \(PPF_M\) is precomputed. It counts the number of times a discretized PPF feature appears on \(M\). The PPF for any two points on \(M_O\) is given by:
where $n_1$ and $n_2$ are point normals and $d$ is the distance between the points. This avoids outliers from $P_O$. For sampling one base, 4 points are sampled incrementally by using a heuristic score associated with every point on the point cloud $P_O$. The heuristic score follows an exponential distribution of the Euclidean Distance Transform of each point, which is computed from the hand’s signed distance field $SDF$:

$$
\pi(p_i) \approx 1 - \exp(-\lambda SDF(p_i; x_H)).
$$

where $\pi(p_i)$ returns a point’s probability to be sampled. The probability distribution of all the points on the object cloud $P_O$ are normalized and denoted as $\pi$. Points further away from the hand are more likely to belong to the object and are prioritized. To balance exploitation and exploration, a discounting factor $\gamma = 0.5$ decays the heuristic when a point is sampled. The discounting generates more dispersed and promising pose hypotheses.

The sampling ensures that the 4 points are co-planar given a small threshold (Line 18). Base sampling is repeated until a desired number of bases is achieved. Given a base $B$, its congruent set on the object model is retrieved by hyper-sphere rasterization [2]. Alignment between the matching bases can be solved in a least square manner [2]. This returns a set of object pose hypotheses along with their LCP score. Base sampling and alignment are executed in parallel.

The large number of pose candidates generated often contains many incorrect or redundant poses. Clustering in SE(3) is performed to group together similar poses and reduce the size of the hypotheses set. Similar to prior work [8], a fast and effective technique is adapted for this step: a round of coarse grouping is performed in $R^3$ via Euclidean Distance Clustering. Then, each group is split by clustering according to the minimal geodesic distance along $SO(3)$:

$$PPF(p_1, p_2) = (\|p_1p_2\|_2, \angle(n_1,d), \angle(n_2,d), \angle(n_1,n_2))$$
Algorithm 10: SAMPLE_ONE_BASE \((P_O, \pi, PPF_M)\)

1. \(b_1 \leftarrow\) sample a point from \(P_O\) according to \(\pi\);
2. \(B \leftarrow \{b_1\}\);
3. \textbf{for} \(p \in P_O\) \textbf{do}
4. \hspace{1em} \(f_1 \leftarrow PPF(p, b_1)\);
5. \hspace{1em} \textbf{if} \(PPF_M[f_1] = \emptyset\) \textbf{then}
6. \hspace{2em} \(\pi(p) \leftarrow 0\);
7. \hspace{1em} \textbf{for} \(i \leftarrow 0\) \textbf{to} \text{max}_\text{iter} \textbf{do}
8. \hspace{2em} \(b_2, b_3 \leftarrow\) sample two different points from \(P_O\) according to the updated distribution \(\pi\);
9. \hspace{3em} \(\pi(b_2) \leftarrow \gamma \pi(b_2)\);
10. \hspace{3em} \(\pi(b_3) \leftarrow \gamma \pi(b_3)\);
11. \hspace{3em} \(f_{23} \leftarrow PPF(b_2, b_3)\);
12. \hspace{2em} \textbf{if} \(PPF_M[f_{23}] \neq \emptyset\) and \(\angle(b_1b_2b_3) > \delta\) \textbf{then}
13. \hspace{3em} \(B \leftarrow B \cup \{b_2, b_3\}\);
14. \hspace{2em} \textbf{break} ;
15. \hspace{1em} \textbf{for} \(i \leftarrow 0\) \textbf{to} \text{max}_\text{iter} \textbf{do}
16. \hspace{2em} \(b_4 \leftarrow\) sample a point from \(P_O\) according to the updated distribution \(\pi\);
17. \hspace{3em} \(\pi(b_4) \leftarrow \gamma \pi(b_4)\);
18. \hspace{3em} \textbf{if} \(\text{distance}(\text{plane}(b_1, b_2, b_3), b_4) > \varepsilon\) \textbf{then}
19. \hspace{4em} \textbf{continue} ;
20. \hspace{3em} \(f_{24} \leftarrow PPF(b_2, b_4), f_{34} \leftarrow PPF(b_3, b_4)\);
21. \hspace{3em} \textbf{if} \(PPF_M[f_{24}] \neq \emptyset\) and \(PPF_M[f_{34}] \neq \emptyset\) \textbf{then}
22. \hspace{4em} \(B \leftarrow B \cup b_4\);
23. \hspace{2em} \textbf{break} ;
24. \hspace{1em} \textbf{return} \(B\);

\[
d(R_1, R_2) = \arccos\left(\frac{\text{trace}(R_1^T R_2) - 1}{2}\right).
\]

Different from prior work [8], however, rather than using K-means, which can be computationally expensive, the new hypotheses are formed by the poses with the highest LCP score per cluster and refined by Point-to-Plane ICP [138]. After ICP, some candidates may converge to the same pose and are merged. The top \(k\) hypotheses (empirically set to 100) with the highest LCP score are kept to improve computational efficiency.
### Table 7.11: Comparison on simulation dataset. For the table, +HS implies using the proposed PSO hand pose estimation to remove the hand related cloud from the scene, +ICP implies applying Point-to-Plane ICP for pose refinement.

<table>
<thead>
<tr>
<th>Method</th>
<th>Avg. Recall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super4PCS [53] + HS</td>
<td>71.58</td>
</tr>
<tr>
<td>Super4PCS [53] + HS + ICP</td>
<td>78.83</td>
</tr>
<tr>
<td>DOPE [14] + ICP</td>
<td>31.88</td>
</tr>
<tr>
<td>DOPE [14] + HS + ICP</td>
<td>35.58</td>
</tr>
<tr>
<td>DOPE [14] + HS + ICP</td>
<td>55.40</td>
</tr>
<tr>
<td>AAE* [60] + ICP</td>
<td>57.77</td>
</tr>
<tr>
<td>AAE* [60] + HS + ICP</td>
<td>69.85</td>
</tr>
<tr>
<td>AAE* [60] + HS + ICP</td>
<td>78.06</td>
</tr>
<tr>
<td>OURS</td>
<td>95.33</td>
</tr>
</tbody>
</table>

#### 7.2.4 Pose Hypothesis Pruning and Selection

Physical reasoning is leveraged to further prune false hypotheses via collision checking and scene-level occlusion reasoning. Physical consistency is imposed by checking if the object model collides beyond certain depth with the estimated hand state, or if the object is located above certain distance from the hand mesh surface, indicating that the hand is not touching the object. This process can be performed efficiently by utilizing the hand state and its SDF.

Ambiguities might still arise due to several pose candidates achieving similar LCP score with the object under high occlusions. Any (non-corrupted) observation of a non-zero depth indicates that there is nothing between the observed point and the camera, up to some noise threshold and barring sensor error [139]. This scene-level reasoning is adapted by comparing the accumulated pixel-wise discrepancy between the observed depth image and the rendered one (computed via OpenGL using both the estimated object pose and hand state). Based on this rendering score, the top 1/3rd of pose hypotheses are retained. The final optimal pose is selected from this set according to the highest LCP.

#### 7.2.5 Experimental Setup

This section evaluates the proposed approach and compares against state-of-the-art single-image pose estimation methods on in-hand objects. Note the difference with tracking methods [140], since here the 6D object pose is recovered from a single static image without dependency on previous frames. To the best of the authors’ knowledge, there are no relevant datasets in the literature beyond those for objects in human hands [141]. A benchmark...
dataset is developed that includes both simulated and real world data for in-hand object pose estimation with adaptive hands and will be released publicly.

The setup consists of a robot manipulator (Yaskawa Motoman) and a Yale T42 adaptive hand (Fig. 7.9), which was 3D printed based on open-source designs. Objects considered for in-hand manipulation were picked to evaluate the robustness of estimation. As shown in Fig. 7.12, the selected set is a mix of objects: with and without texture or geometric features.

Figure 7.12: Mesh of objects used: A cylinder with diameter $0.035 \, m$ and length $0.064 \, m$, an ellipsoid with length $0.064 \, m$, a cuboid with side length $0.03 \, m$ and length $0.064 \, m$, an industrial object #3 from T-LESS dataset [142], a mustard bottle and tomato soup can from YCB dataset [73]. Right 2 images: Yale T42 adaptive hands painted in blue and white.

All experiments are conducted on a standard desktop with Intel Xeon(R) E5-1660 v3@3.00GHz processor. For the comparison to deep learning methods, neural network inference is performed on a NVIDIA Tesla K40c GPU.

### 7.2.6 Evaluation Metric

The recall for pose estimation is measured based on the error given by the $\text{ADI}$ metric [36], which measures the average of point distances between poses $T_1$ and $T_2$ given an object mesh model $M$:

$$
\epsilon_{\text{ADI}}(T_1, T_2) = \text{avg}_{p_1 \in M} \min_{p_2 \in M} ||T_1(p_1, M) - T_2(p_2, M)||_2,
$$

where $T(p, M)$ corresponds to point $p$ after applying transformation $T$ on $M$. Given a ground-truth pose $T^g$, a true positive is a returned pose $T$ that has $\epsilon_{\text{ADI}}(T, T^g) < \epsilon$, where $\epsilon$ is a tolerance threshold. $\epsilon$ is set to $5 \, mm$ in all experiments except in recall curves, to evaluate the applicability of different methods for precise in-hand manipulation scenarios.
7.2.7 Simulation Dataset and Results

Simulated RGB-D data were generated by placing a virtual camera at random poses around the model of the hand. Poses are sampled from 648 view points on spheres of radius 0.3 to 0.9 m centered at the hand. To generate each data point, an object is placed at a random pose between the fingers. The two articulated-fingers are closed randomly until they touch the object, verified by a collision checker. Physical parameters, such as friction, gravity or any grasping stability metric are deliberately not employed since this work aims at any-time single image 6D object pose estimation during the entire within-hand manipulation process, in which a stable grasp is not always a true assumption. By randomizing the object pose relative to the hand, the dataset is able to cover various in-hand object poses that can occur during an in-hand manipulation process. For the adaptive hand, two colors are chosen. The blue hand differs from any object color used in the experiments whereas the white hand resembles texture-less objects and evaluates robustness to lack of texture. In addition to the RGB-D data, ground-truth object pose and semantic segmentation images are also obtained from the simulator. For each combination of the 6 objects and the 2 adaptive hands, 1000 data points are generated, resulting in 12000 test cases.

Fig. 7.11 reports the recall for pose estimation on the synthetic dataset. When Super4PCS is directly applied to the entire point cloud, outlier points that do not belong to the object are often sampled, leading to poor results (5.83 %). On introducing the proposed PSO hand state estimation (HS) and thereby eliminating the hand points from the scene, points belonging to the object are more likely to be sampled, which dramatically improves the performance of (Super4PCS+HS). Recent state-of-the-art learning-based approaches are also evaluated. DOPE [47] trains a neural network to predict 3D bounding-box vertices projected on the image and recovers 6D pose from them via Perspective-N-Point (PnP), which has shown to outperform PoseCNN [48] on the YCB dataset. To eliminate the domain gap from the scope of evaluation, the training and test data were generated in the same simulator and domain randomization was utilized as suggested [47]. AAE [54] is another learning-based method that trains an autoencoder network to embed object 3D orientation information using extensive data augmentation.
and domain randomization techniques. It has been shown to be successfull on textureless objects and achieved state-of-art results on the T-LESS dataset [142]. This approach is only able to predict 3D orientation. The translation is based on the output of another object detection network. For the scope of this evaluation *ground-truth* bounding-box were provided as input to AAE [54].

A dramatic performance improvement is observed for all methods, when the proposed *PSO* hand state estimation is utilized to remove the hand related point cloud from the scene. This proves the significance of additionally estimating the robot hand state for in-hand 6D object pose estimation.

<table>
<thead>
<tr>
<th>Method</th>
<th>Modality</th>
<th>cylinder</th>
<th>cuboid</th>
<th>ellipse</th>
<th>tless3</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super4PCS [2] + HS</td>
<td>Depth</td>
<td>52.49</td>
<td>43.85</td>
<td>62.64</td>
<td>62.64</td>
<td>55.41</td>
</tr>
<tr>
<td>Super4PCS [2] + HS+ICP</td>
<td>Depth</td>
<td>70.51</td>
<td>43.85</td>
<td>54.81</td>
<td>78.49</td>
<td>61.92</td>
</tr>
<tr>
<td>AAE* [54]</td>
<td>RGB</td>
<td>11.19</td>
<td>8.56</td>
<td>15.92</td>
<td>40.38</td>
<td>19.01</td>
</tr>
<tr>
<td>AAE* [54] + ICP</td>
<td>RGBD</td>
<td>43.39</td>
<td>22.99</td>
<td>27.35</td>
<td>55.85</td>
<td>37.40</td>
</tr>
<tr>
<td>AAE* [54] + HS+ICP</td>
<td>RGBD</td>
<td>41.02</td>
<td>29.41</td>
<td>29.80</td>
<td>81.89</td>
<td>45.53</td>
</tr>
<tr>
<td>OURS</td>
<td>Depth</td>
<td>87.12</td>
<td>72.19</td>
<td>80.82</td>
<td>93.96</td>
<td>83.52</td>
</tr>
</tbody>
</table>

Table 7.1: Recall percentage (\(e_{ADI} < 5 \text{mm}\)) on real data: +HS means using the proposed PSO hand state estimation to remove the hand’s point cloud, +ICP means applying Point-to-Plane ICP at the end for pose refinement. AAE* [54] is provided a *ground-truth* bounding-box.
7.2.8 Real Dataset and Results

The real dataset contains 986 snapshots of 2 Yale T42 hands holding 4 types of objects including cylinder (295), ellipse (239), cuboid (187) and tless3 (265). All the objects and the adaptive hands are 3D printed. Similar to the setting in simulation, the adaptive hands are painted in two colors: blue-green and white. The images are collected with an Intel RealSense SR300 RGB-D camera and the ground-truth poses are manually annotated using a GUI developed by the authors. Before each image is taken, objects are grasped randomly and the adaptive hand performs a random within-hand manipulation. Due to the small size of objects relative to the hand, severe occlusions occur frequently, as exhibited in Fig. 7.13.

![Figure 7.14: Pose recall of [4] on the real dataset. As the approach requires initialization, it is evaluated over perturbations on the ground-truth pose.](image)

Table 7.1 presents results on real data. Given the large appearance gap between synthetic training data and real scenarios, and the presence of textureless objects, the performance of DOPE does not translate well, and thereby was dropped from the table. AAE [54] was robust to some of these challenges and given the ground-truth bounding-boxes, it could predict the correct rotation in some cases. An additional related work [4] was evaluated on real data. It was developed to perform pose estimation for in-hand objects during robot manipulation. It assumes the initial object pose does not change much upon grasping and serves as an initialization for ICP. To evaluate this approach, pose initialization is provided by perturbing the ground-truth pose. Fig. 7.14 shows how the performance of this approach varies with the perturbation. Our proposed approach outperforms the best-case (small perturbation) of [4] even though pose initialization is not provided to our system.
Table 7.2: Ablation study of critical components in our system. Results are averaged across the entire real dataset. Baseline refers to random base sampling on the entire scene cloud.

Fig. 7.13 exhibits examples of the output from the proposed approach on real data where severe occlusions occur and additional challenge is introduced by virtue of the noise in consumer-level depth sensor. Table 7.2 shows the ablation study where the recall percentage for the object pose ($e_{ADI} < 5\text{mm}$) is measured by incrementally adding the critical proposed components.

### 7.3. In-hand Pose Tracking

Tracking the 6D pose of objects in video sequences is important for robot manipulation in domains, such as logistics and service robotics. At the same time, however, these tasks frequently involve significant occlusions, which complicate the tracking process. Machine learning solutions are challenged because ground truth data and labels are difficult to collect for 6D poses. Furthermore, incremental error drift in long-term tracking often accumulates enough to necessitate re-initialization of the tracked object’s pose. This work proposes a data-driven optimization approach for long-term, 6D pose tracking. It aims to identify the optimal relative pose given the current RGB-D observation and a synthetic image conditioned on the previous best estimate. A key contribution in the context of this framework is a novel neural network architecture, which can be trained only with synthetic data but works effectively over real images. An important feature for the effectiveness of the approach corresponds to the appropriate representation of 3D orientations via Lie Algebras and its training loss function. Comprehensive experiments over benchmarks - existing ones as well as a new dataset with significant occlusions related to object manipulation - show that the proposed approach achieves consistently robust estimates and outperforms alternatives, even though they have been trained with real images. The approach is also the most computationally efficient among the alternatives (at 90.9Hz). Ablation studies have been performed to highlight the impact of important components of the approach.
Figure 7.15: **Left:** Performance w.r.t. computation time evaluated on the YCB-Video dataset according to the area under the curve (AUC) metric for the ADD and ADD-S objectives [48]. The proposed approach is able to perform more accurate tracking while being significantly faster than alternatives. **Right:** Pose predicted by se(3)-TrackNet without any re-initialization, which is able to recover from complete occlusion.

### 7.3.1 Problem Statement

The objective of this work is to compute the 6D pose of an object $T_t \in SE(3)$ at any time $t > 0$, given as input:

- a 3D CAD model of the object,
- its initial pose, $T_0 \in SE(3)$ computed by any single-image based 6D pose estimation technique, and
- sequence of RGB-D images $O_t, \tau \in \{0, 1, \ldots, t - 1\}$ from previous time stamps and the current observation $O_t$.

This work proposes a data-driven optimization technique to track the object pose over RGB-D image sequences. The cost function for the optimization is encoded and learned by a novel neural network architecture, trained with only synthetically generated data. In every time step, the proposed approach, computes a residual over the pose computed for the object model in the previous frame as indicated in Fig. 2. This is different from traditional tracking approaches that accumulate transformations between consecutive frames and are prone to drifts. The details of this formulation, the neural network architecture and the data generation pipeline follow.

### 7.3.2 Learning to Track Along $SE(3)$ Manifolds via Calibrating Image Residuals

Optimization techniques in this domain operate over cost functions defined over a pair of images $I_1, I_2$ parametrized by their poses $\bar{\xi}, \xi$ that measure the discrepancy $\varepsilon$ between the features
Figure 7.16: Pipeline: At any given time \( t \) the current observation \( O_t \) and the render for the object model \( R_{t-1} \) based on the previously computed pose \( \xi_{t-1} \) are passed to the proposed network. The network computes the relative pose \( \Delta \xi_{t-1} \) which is then propagated forward to the next step.

extracted from the images:

\[
\varepsilon = \rho(\phi_i(\xi) - \phi_E(\xi))
\]

where \( \rho \) is a predefined robust loss function, and \( \phi(\cdot) \) can be direct pixel intensity values, such as in [143, 144], point-to-point discrepancy [58] as well as their variations [59], pre-designed features [145, 146] or the combinations from any of the above [147, 148].

Given the current observation \( O_t \), and the pose computed in the previous timestamp \( \xi_{t-1} \), the goal of this work is to find a relative transformation \( \Delta \xi \) that takes the object from \( \xi_{t-1} \) to the pose captured by the current observation. This can be formulated as an optimization problem. Let \( R \) denote the image corresponding to the rendering of the object model at the given pose. Then, the optimal relative transform can be defined as:

\[
\Delta \xi^* = \arg\min_{\Delta \xi} \{ \rho(\phi_O(\xi) - \phi_R(\xi_{t-1} \oplus \Delta \xi)) \}
\]

A general approach to solve this is by performing Taylor Expansion around \( \xi \), which allows to rewrite the equation as \( \phi_R(\xi_{t-1} \oplus \Delta \xi) = \phi_R(\xi_{t-1}) + J(\xi_{t-1})\Delta \xi \), where \( J \) is the Jacobian matrix of \( \phi_R \) w.r.t. \( \xi \). Now \( \xi \) is locally parametrized in its tangent space, specifically \( \xi = (t, w)^T \in \)
Figure 7.17: Network architecture for $se(3)$-TrackNet. It takes as input RGB-D images corresponding to current observation and a rendering of object model at previous timestamp in two separate feature encoders $\phi_B$ and $\phi_A$ respectively. Both the inputs are synthetic during training while at test time, input to $\phi_B$ is a real image. The output from the encoders are concatenated and used to predict relative pose between the two images, with decoupled translation and rotational component. The output transformation is represented by Lie algebra as described in section 7.3.2

$se(3)$ in this case, such that its exponential mapping lies in the Lie Group $\Delta T = \exp(\Delta \xi) = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in SE(3)$, where $R = I_{3 \times 3} + \frac{|w|}{|w|} \sin(|w|) + \frac{|w|^2}{|w|^2} (1 - \cos(|w|))$ and $[w]_\times$ is the skew-symmetric matrix.

In the case of $L_2$ loss function without loss of generality:

$$\Delta \xi = (J^T J)^{-1} J^T \| (\phi_O(\xi_t) - \phi_R(\xi_{t-1} \boxplus \Delta \xi)) \|.$$  

Solving the expression by explicitly deriving the Jacobian matrix and iteratively updating often requires a formalized cost function or the features extracted from the observations to be differentiable w.r.t. $\xi$, and appropriate choice of a robust cost function and hand-crafted features. Another problem arises when different modalities are involved. In such cases, another hyper-parameter controlling the importance of each modality (e.g., RGB-D) has to be introduced and could become non-trivial to tune for all different scenarios.

Instead, this work proposes a novel neural network architecture that implicitly learns to calibrate the residual between the features extracted from the current observation and the rendered image conditioned on previous pose estimate so as to resolve their optimal relative transform in the tangent space $\Delta \xi \in se(3)$. 
7.3.3 Neural Network Design

The proposed neural network architecture is shown in Fig. 7.17. The network takes as input a pair of images $I^{t-1}$ which is rendered from previous pose estimation and $I^t$ which is the current observation. The images are a concatenation of RGB and depth channels to generate a 4-channel input. Depth is often available in robotics setup, and provides an additional modality to accurately compute object pose. Nevertheless, it complicates the learning process due to additional domain-gap between synthetic and real depth images and the fact that not all neural network architectures are well suited to encode RGBD features, such as the FlowNetSimple architecture [149] used in [150].

During the train time, both inputs are synthetically generated images $\phi(I^{t-1}_{syn,train}; I^{t}_{syn,train})$, while at test time the current timestamp input is obtained from a real sensor, $\phi(I^{t-1}_{syn,test}; I^{t}_{real,test})$. The proposed network utilizes two separate input branches for $I^{t-1}$ and $I^t$ before sending them into the network. The weights of the feature encoders are not shared.

This is different from the related work [150], where the two images are concatenated into a single input to the network. A shared feature extractor worked in the previous work as both real and synthetic data are available at training time and thus the property of the latent space $\phi(I^{t-1}_{syn,train}; I^{t}_{real,train})$ could still be partly preserved when tested on real world test scenarios $\phi(I^{t-1}_{syn,test}; I^{t}_{real,test})$. This representation does not generalize to the present scenario with only synthetic training data.

The latent space features trained on purely synthetic data are denoted by $\phi_A(I^{t-1}_{syn,train})$ and $\phi_B(I^{t}_{syn,train})$. When tested on real world data, the latent space features are $\phi_A(I^{t-1}_{syn,test})$ and $\phi_B(I^{t}_{real,test})$. By this latent space decoupling, domain gap reduces to be between $\phi_B(I^{t}_{syn,train})$ and $\phi_B(I^{t}_{real,test})$ while $\phi(I^{t-1}_{syn,train})$ and $\phi(I^{t-1}_{syn,test})$ can be effortlessly aligned between the training and test phase without the need for tackling the domain gap problem.

A relative transform can then be predicted by the network by end-to-end training. The transformation is represented by lie algebra as $\Delta \xi = (t, w)^T \in se(3)$, where the prediction of $w$ and $t$ are disentangled into two separate branches and trained by $L_2$ loss:

$$L = \lambda_1 ||w - \bar{w}||_2 + \lambda_2 ||t - \bar{t}||_2$$
Figure 7.18: Comparison of Domain Randomization against PPDR (Physically Plausible Domain Randomization). **Top Left:** Domain Randomization tends to directly render using a sampled object pose. Notice the penetration between objects that differs from natural occlusion scenarios and could introduce undesired bias to the depth training data. **Top Right:** In PPDR, a randomly sampled pose serves as an initial configuration for a physics simulation process, and rendering is performed over the final object pose. The domain invariant, penetration-free property can then be effectively aligned between synthetic and real domains. **Bottom:** Examples of synthetic scenes generated with PPDR.

where $\lambda_1$ and $\lambda_2$ has been simply set to 1 in experiments. Given $\Delta \xi$, the current pose estimate is computed as $T^t = \exp(\Delta \xi) \cdot T^{t-1}$, as described in Sec. 7.3.2. The effectiveness of the representation and loss function have been validated in experiments compared against alternatives, such as Shape-Match Loss employed in prior works [48, 150, 49] or rotation representations, such as quaternions.

### 7.3.4 Synthetic Data Generation via Physically Plausible Domain Randomization

The purpose of domain randomization is to provide enough simulated variability at training time, such that at test time, the model is able to generalize to real-world data [53]. Prior work implements the idea of domain randomization by randomly changing the number of objects, poses, textures, lighting, etc, where object poses are usually directly sampled from some predetermined distribution [53, 48, 47]. Although it is non-trivial to align some complex physical properties between the simulator and real world, such as lighting and camera properties, the current work argues that certain physical properties, for instance gravity and collision, can be effortlessly preserved. This makes domain invariant features more tractable to be captured by the neural network. This could be especially beneficial in some cases when (1) computational resources are limited and large-scale simulation data generation is not feasible, or (2) depth
modality is employed and unrealistic object penetration, which never occurs in real world, could introduce undesired bias to the training data.

This work therefore leverages the complementary attributes of domain randomization and physically-consistent simulation \cite{6} for the synthetic data generation process. The goal is to combine the two ideas such that the synthetic training data holds a diverse distribution for the network to generalize to the target domain with different environments, while being more data-efficient. We refer to this idea as PPDR (Physically Plausible Domain Randomization). More concretely, object poses are initialized randomly where collision between objects or distractors could occur, which is then followed by a number of physics simulation steps so that objects are separated or fall onto the table without collision. Other complex or intractable physical properties such as lighting, number of objects, distractor textures are randomized. A comparison between Domain Randomization against PPDR as well as some example synthetic scenes are available in Fig. 7.18.

Once the synthetic image is generated for the entire scene, paired data $I_{syn, train}^{-1}$ and $I_{syn, train}$ are extracted and utilized as the input to the network. $I_{syn, train}^{-1}$ is obtained by cropping the image given the target object’s dimension and zoomed into a fixed resolution $176 \times 176$ before feeding into the network - similar to prior work \cite{150}. $I_{syn, train}^{-1}$ is obtained by randomly sampling a perturbated pose $T_{t-1} = \exp(\begin{bmatrix} w \times & t \\ 0 & 1 \end{bmatrix})$ where the translation’s direction is uniformly sampled and its norm follows a Gaussian distribution $|t| \sim |N(0, \sigma_t)|$. The rotation is locally parameterized in the tangent space $w \in so(3)$ as discussed above and the direction of $w$ is also uniformly sampled, while its norm is sampled from a Gaussian distribution $|w| \sim |N(0, \sigma_w)|, w \in \mathbb{R}^3$, similar to $t$.

The next step within this context is to bridge the domain gap of depth data via bidirectional alignment. Similar to the domain gap that exists between synthetic and real RGB images, discrepancies also arise between synthetic and real depth data, especially for those captured by a commercial-level depth sensor. Although significant effort has been invested in bridging the domain gap in RGB images, such as the Domain Randomization discussed in the above section, there has been less evidence about how the depth domain gap could be resolved in a general way, since it could be partly dependent on the specific depth sensor. Inspired by related work
114

During training time, two additional data augmentation steps are applied to the synthetic depth data $D_{syn,train}^{t}$ at branch B. First, random Gaussian noise is added to the pixels with a valid depth value, which is then followed by a depth-missing procedure by randomly changing each pixel that has a valid depth value to invalid so as to resemble a real corrupted depth image captured by commercial-level depth sensors. In contrast, during test time, a bilateral filtering is carried out on the real depth image so as to smooth sensor noise and fill holes to be aligned with the synthetic domain.

7.3.5 Experiment

This section evaluates the proposed approach and compares against state-of-the-art 6D pose tracking methods as well as single-image pose estimation methods on a public benchmark. It also introduces a new benchmark developed as part of this work, which corresponds to robot manipulation scenarios. This new benchmark will be released publicly. Extensive experiments are performed over diverse object categories and various robotics manipulation scenarios (moving camera or moving objects). Both quantitative and qualitative results demonstrate the advantages of the proposed method in terms of accuracy and speed, while using only synthetic training data. Except for training, all experiments are conducted on a standard desktop with Intel Xeon(R) E5-1660 v3@3.00GHz processor. Neural network training and inference are performed on a NVIDIA RTX 2080 Ti GPU and NVIDIA Tesla K40c GPU respectively.

7.3.6 Datasets

**YCB-Video Dataset** This dataset captures 92 RGB-D video sequences over 21 YCB Objects arranged on table-tops. A variety of objects appear with or without texture, having different shapes and dimensions, thereby exhibiting challenges to both RGB and depth sensing modalities. Objects are annotated with ground-truth 6D object poses. The evaluation closely follows the protocols adopted in comparison methods and reports the AUC (Area Under Curve) results on the keyframes in 12 video test sequences evaluated by the metrics of $ADD = \frac{1}{m} \sum_{x \in M} ||Rx + T - (\hat{R}x + \hat{T})||$ which performs exact model matching, and $ADD - S = \frac{1}{m} \sum_{x_1 \in M} \min_{x_2 \in M} ||Rx_1 + T - (\hat{R}x_2 + \hat{T})||$ designed for evaluating symmetric
objects, of which the matching between points can be ambiguous for some views.

Although this dataset contains pose annotated training and validation data collected in real world, the proposed approach does not use any of them but is trained only on separately generated synthetic data.

**YCBInEOAT Dataset** There have been several public benchmarks [48, 154] where videos are collected by placing the objects statically on a table-top while a camera is moved around to imitate a 6D object pose tracking scenario. This can be limiting for evaluating 6D pose tracking since in a static environment, the entire image can be leveraged to solve for the trajectory of the camera, from which object pose can be inferred [145, 155, 142]. Additionally, in such scenarios, extreme object rotations, such as out-of-image-plane flipping are less likely to happen than a free moving object in front of the camera. Thus, exclusive evaluation on such datasets cannot entirely reflect the attributes of a 6D object pose tracking approach. For these reasons, other datasets [141, 156, 157] collected video sequences where objects are manipulated by a human hand. Nevertheless, human arm and hand motions can greatly vary from those of robots. The lack of forward-kinematics information also hinders a sufficiently precise pose estimation required in robot manipulation tasks.

Therefore, in this work a novel dataset, referred to as ”YCBInEOAT Dataset”, is developed in the context of robotic manipulation, where various robot end-effectors are included: a vacuum gripper, a Robotiq 2F-85 gripper, and a Yale T42 Hand [134]. The manipulation sequences consider 5 YCB objects, given their widely accessibility. The data collection setup and objects are shown in Fig. 7.19. Each video sequence is collected from a real manipulation performed with a dual-arm *Yaskawa Motoman SDA10f*. In general, there are 3 types of manipulation tasks performed: (1) single arm pick-and-place, (2) within-hand manipulation, and (3) pick to hand-off between arms to placement. RGB-D images are captured by an *Azure Kinect* sensor mounted statically on the robot with a frequency of 20 to 30 Hz. Forward kinematics for the manipulator are recorded at approximately 100 Hz. Similar to the YCB-Video, ADD and ADD-S metrics are adopted for evaluation. Ground-truth 6D object poses in camera’s frame have been accurately annotated *manually* for each frame of the video. The extrinsic parameters of the camera in the frame of the robot has been obtained by a calibration procedure. This dataset will become publicly available for future benchmarking.
without collisions or fallen onto the table. For physics simulation, which is terminated after 50 steps to ensure objects have been separated.

The synthetic data generation pipeline is implemented in Blender [101]. To render images, the camera’s pose is randomly sampled from a sphere of radius between 0.6 to 1.3 m, followed by an additional rotation along camera z-axis sampled between 0 to 360°. The number of external lighting sources (lamps) is sampled within 0 to 2 with varying poses. The strength and color of the environment and the lamps are randomized. Object poses are randomly initialized, followed by physics simulation, which is terminated after 50 steps to ensure objects have been separated without collisions or fallen onto the table. For YCB Video, table textures are randomly selected.

Table 7.3: Comparing the performance of se(3)-TrackNet (Gray) with state-of-the-art techniques on YCB Video. The approach significantly outperforms all competing approaches over the ADD metric which considers semantic information during pose evaluation. It also achieves the highest success rate over the ADD-S metric both in cases of initialization with the ground-truth pose and when initialized with the output of PoseCNN [45] (rightmost two columns)

7.3.7 Implementation Details

The synthetic data generation pipeline is implemented in Blender [101]. To render images, the camera’s pose is randomly sampled from a sphere of radius between 0.6 to 1.3 m, followed by an additional rotation along camera z-axis sampled between 0 to 360°. The number of external lighting sources (lamps) is sampled within 0 to 2 with varying poses. The strength and color of the environment and the lamps are randomized. Object poses are randomly initialized, followed by physics simulation, which is terminated after 50 steps to ensure objects have been separated without collisions or fallen onto the table. For YCB Video, table textures are randomly selected.
from [159]. For YCBInEOAT, tables are removed and background is replaced by images captured in the same location, where the data were collected. For each pair $I_{syn,train}^{t-1}$ and $I_{syn,train}^t$, their relative transformation $T_{t-1}^t$ is sampled following a Gaussian distribution as described in Sec. 7.3.3 where $\sigma_l$ and $\sigma_w$ are empirically set to 2 cm and 0.262 rad ($= 15^\circ$) respectively. 200k data points (image pairs) are used for training the training set. The network is trained with Adam optimizer for 300 epochs with a batch size of 200. Learning rate starts from 0.001 and is scaled by 0.1 at epochs 100 and 200. Input RGB-D images are resized to 176 × 176 before sending to the network. Data augmentations including random HSV shift, Gaussian noise, Gaussian blur are added only to $I_{syn,train}^t$. Additional depth-missing corruption augmentation is applied to $D_{syn,train}^t$ as described in Sec. 7.3.3 by a missing percentage between 0 to 0.4. For both training and inference, rendering of $I^t$ is implemented in C++ OpenGL.

7.3.8 Results on YCB-Video

Table 7.3 and Fig. 7.15 present the evaluation over the YCB-Video dataset. The proposed approach is compared with other state-of-art 6D object pose detection approaches [48, 47, 150, 49] and 6D pose tracking approaches [153, 150, 141, 158], where publicly available source code is used to evaluate [141, 158], while other results are adopted from the respective publications. Among the compared tracking methods, PoseRBPF [153] is the only one that is initialized using predicted poses from PoseCNN [48]. For a fair comparison, two additional

https://github.com/bayesian-object-tracking/dbot
experiments using exactly the same initial pose as PoseRBPF are performed and presented in the rightmost two columns of Table 7.3 one is without any re-initialization, and the other allows re-initialization by PoseCNN twice after heavy occlusions same as in PoseRBPF. The prior work [150] originally proposed to refine the pose output from any 6D pose estimation detection method, but also extends to RGB-based tracking. It has to be re-initialized by PoseCNN when the last 10 frames have an average rotation greater than 10 degrees or an average translation greater than 1 cm, which happens every 340 frames on average as reported [150]. The initial pose is the ground-truth one.

In practice, re-initialization can be quite expensive in robotics applications given the slower running speed of 6D pose detection approaches, which can interrupt and adversely affect other components of the system, such as planning and control. In contrast, the proposed network performs long-term, accurate tracking with less frequent or even no re-initialization while trained using only on synthetic data. Additionally, the proposed method does not require any hyper-parameter tuning or hand-crafted features but generalizes to different lighting conditions and a variety of objects with different properties, such as scissors, and clamps, which are challenging to alternatives. Another important aspect is that when the ADD metric is used for evaluation, the proposed approach is able to achieve 93.05% success, outperforming all comparison methods by a large margin. This can be attributed to its implicitly learnt residual estimator, which not only captures the discrepancy of geometry but also semantic textures by considering both the RGB and Depth modalities.

7.3.9 Results on YCBInEOAT-Dataset

Table 7.4 shows the quantitative results evaluated by the area under the curve for ADD and ADD-S on the developed YCBInEOAT dataset. On this benchmark, the tracking approaches with publicly available source code could be directly evaluated [158, 141]. Pose is initialized with ground-truth in the first frame and no re-initialization is allowed. Example qualitative results are demonstrated by Fig. 7.20 where a vacuum gripper is performing a pick-and-place manipulation task. Due to the non-prehensile nature of the grasp, abrupt motions and extreme

---
2We thank the authors for providing the initial pose they used in the original paper [153]
rotations are introduced which are challenging for 6D object pose tracking. Nevertheless, the proposed approach is sufficiently robust during such undesirable motions and is able to provide long-term reliable pose estimation until the end of the manipulation.

### 7.3.10 Ablation Study

An ablation study investigates the importance of different modules of the proposed approach. It is performed for the large clamp object from the YCB-Video dataset and is presented in the accompanying table. The initial pose is given by ground-truth and no re-initialization is allowed. **No physics** implies that domain randomization is employed during synthetic data generation without any physics simulation. For **No depth**, the depth modality is removed in both training and inference stage to study its importance. **Shared encoder** means the two feature encoders for \( I^t \) and \( I^{t-1} \) share the same architecture and weights. This corresponds to the one used for \( I^{t-1} \) in the original setup. **Quaternion** implements the rotation by quaternion representation \( q = (x, y, z, w) \), where \( w = \sqrt{1 - x^2 - y^2 - z^2} \) is forced to be non-negative so as to avoid ambiguity of \( q \) and \( -q \). The network is trained by \( L_2 \) loss over the representation. **Shape-Match Loss** is a popular loss function in 6D pose estimation task that does not require the specification of symmetries [48]. It loses track, however, of the object very early in the current setting.
Figure 7.22: Overview of our approach for 6D pose estimation at inference time. This figure shows the pipeline for the drill object of the YCB-video dataset [48]. A deep learning model is trained with weakly annotated images. Extracted class-specific heatmaps, along with 3D models and the depth image, guide the Stochastic Congruent Sets (StoCS) method [7] to estimate 6D object poses. Further details of the network are available in Section 7.4.1.

### 7.4. Pose Estimation with Weakly Labeled images

This section presents a weak object detector that enables localizing objects and estimating their 6D poses in cluttered and occluded scenes. To minimize the human labor required for annotations, the proposed detector is trained with a combination of synthetic and a few weakly annotated real images (as little as 10 images per object), for which a human provides only a list of objects present in each image (no time-consuming annotations, such as bounding boxes, segmentation masks and object poses). To close the gap between real and synthetic images, we use multiple domain classifiers trained adversarially. During the inference phase (Fig. 7.22), the resulting class-specific heatmaps of the weak detector are used to guide the search of 6D poses of objects. Our proposed approach is evaluated on several publicly available datasets for pose estimation. We also evaluated our model on classification and localization in unsupervised and semi-supervised settings. The results clearly indicate that this approach could provide an efficient way toward fully automating the training process of computer vision models used in robotics.

#### 7.4.1 Proposed Approach

We present here our approach to object localization and 6D pose estimation. It is trained using a mix of synthetic and real images and only requires weak annotations (only class-presence)
in both domains. Figure 7.23 depicts an overview of our proposed system. It comprises i) a ResNet-50 model pre-trained on ImageNet as a feature extractor (green), ii) a weak classifier inspired from the WILDCAT model [160] (blue), iii) the Stochastic Congruent Sets (StoCS) for 6D pose estimation (red) [7], and iv) the MADA domain adaptation network to bridge the gap between synthetic and real data. During the inference phase, the domain adaptation part of the network is discarded. Given a test image, class-specific heatmaps are generated by the network. These heatmaps indicate the most probable locations of each object in the image. This probability distribution is then fed to StoCS, a robust pose estimation algorithm that is specifically designed to deal with noisy localization. To force the feature extractor to extract similar features for both synthetic and real images, a MADA module (described below) is employed. MADA’s purpose is to generate gradients during training (via a reversal layer) in order to improve the generalization capabilities of the feature extractor.
7.4.2 Synthetic Data Generation

For synthetic data generation, we used a modified version of the SIXD toolkit\footnote{https://github.com/thodan/sixd_toolkit}. This toolkit generates color and depth images of 3D object models rendered on black backgrounds. Virtual camera viewpoints are sampled on spheres of different radii, following the approach described in \cite{161}. We extended the toolkit with the functionality of rendering more than one object per image, and also used random backgrounds taken from the LSUN dataset \cite{162}. Similarly to recent domain randomization techniques \cite{53}, we observed from our experiments that these simple modifications help transferring from simulation to real environments where there are multiple objects of interest, occlusions and diverse backgrounds. Figure 7.23 displays some examples of the generated synthetic images that we used to train our network.

7.4.3 Weakly Supervised Learning with WILDCAT

The images used for training our system are weakly labeled: only a list of object classes present in the image is provided. In order to recover localization from such weak labels, we leverage the WILDCAT architecture \cite{160}. Indeed, WILDCAT is able to recover localization information through its high-level feature map, even though it is only trained with a classification loss. As a feature extractor, we employ a ResNet-50 (pretrained on ImageNet) for which the last layers (global average pooling and fully connected layers) are removed, as depicted in Figure 7.23. The WILDCAT architecture added on top of this ResNet-50 comprises three main modules: a multimap transfer layer, a class pooling layer and a spatial pooling layer. The multimap transfer layer consists of $1 \times 1$ convolutions that extracts $M$ class-specific modalities per class $C$, with $M = 8$ as per the original paper \cite{160}. The class pooling module is an average pooling layer that reduces the number of feature maps from $MC$ to $C$. Then, the spatial pooling module selects $k$ regions with maximum/minimum activations to calculate scores for each class. The classification loss for this module is a multi-label one-versus-all loss based on max-entropy ($MultiLabelSoftMarginLoss$ in PyTorch). The classification scores are then rescaled between 0 and 1 to cooperate with MADA.
7.4.4 Multi-Adversarial Domain Adaptation with MADA

We used the Multi-Adversarial Domain Adaptation (MADA) approach [163] to bridge the “reality gap”. MADA extends the Domain Adversarial Networks (DANN) approach [164] by using one domain discriminator per class, instead of a single global discriminator as in the original version of DANN [164]. Having one discriminator per class has been found to help aligning class-specific features between domains. In MADA, the loss $L_d$ for the $K$ domain discriminators and input $x_i$ is defined as:

$$L_d = \frac{1}{n} \sum_{k=1}^{K} \sum_{x_i \in D_s \cup D_t} L^k_d \left( G^k \left( y^k_i G_f(x_i) \right), d_i \right), \quad (7.2)$$

wherein $i \in \{1, \ldots, n\}$, and $n = n_s + n_t$ is the total number of training images in source domain $D_s$ (synthetic images) and the target domain $D_t$ (real images). $G_f$ is the feature extractor (the same for both domains), $y^k_i$ is the probability of label $k$ for image $x_i$. This probability $y^k_i$ is the output of the weak classifier WILDCAT. $G^k_d$ is the $k$-th domain discriminator and $L^k_d$ is its cross-entropy loss, given the ground truth domain $d_i \in \{\text{synthetic, real}\}$ of image $x_i$. Our global objective function is:

$$C = \frac{1}{n} \sum_{x_i \in D} L_y \left( G_y \left( G_f(x_i) \right), y_i \right) - \lambda L_d, \quad (7.3)$$

where $L_y$ is the classification loss, $L_d$ the domain loss and $\lambda$ has been found to work well with a value of 0.5. The heat-map probability distribution extracted from WILDCAT is used to guide the StoCS algorithm in its search for 6D poses, as explained in the next section.

7.4.5 Pose Estimation with Stochastic Congruent Sets (StoCS)

The StoCS method [7] is a robust pose estimator that predicts the 6D pose of an object in a depth image from its 3D model and a probability heatmap. We employ a min-max normalization on the class-specific heatmaps of the Wildcat network, transforming them into a probability heatmaps $w_{p_i}$, using the per-class minimum ($w_{min}$) and maximum ($w_{max}$) values:

$$\pi_{p_i \rightarrow O_k} = \frac{w_{p_i} - w_{min}}{w_{max} - w_{min}}, \quad (7.4)$$
This generates a heatmap providing the probability $\pi$ of an object $O_k$ being located at a given pixel $p_i$. The StoCS algorithm then follows the paradigm of a randomized alignment technique. It does so by iteratively sampling a set of four points, called a base $B$, on the point cloud $S$ and finds corresponding set of points on the object model $M$. Each corresponding set of four points defines a rigid transformation $T$, for which an alignment score is computed between the transformed model cloud and the heatmap for that object. The optimization criteria is defined as

$$T_{opt} = \arg\max_T \sum_{m_i \in M_k} f(m_i, T, S_k),$$  \hspace{1cm} (7.5)

$$f(m_i, T, S_k) = \pi_k(s^*) \text{ if } |T(m_i) - s^*| < \delta_k.$$  \hspace{1cm} (7.6)

The base sampling process in this algorithm considers the joint probability of all four points belonging to the object in question, given as

$$\Pr(B \rightarrow O_k) = \frac{1}{Z} \prod_{i=1}^{4} \left\{ \phi_{node}(b_i) \prod_{j=1}^{j<i} \phi_{edge}(b_i, b_j) \right\}.$$  \hspace{1cm} (7.7)

where $\phi_{node}$ is obtained from the probability heatmap and $\phi_{edge}$ is computed based on the point-pair features of the pre-processed object model. Thus, the method combines the normalized output of the Wildcat network with the geometric model of objects to obtain base samples which belong to the object with high probability.

In the next two Sections, we demonstrate the usefulness of our approach. First in Section 7.4.6, we quantify the importance of each component (Wildcat, MADA) in order to train a network that generates relevant feature maps from weakly labeled images. In Section 7.4.9, we then evaluate the performance of using these heatmaps with StoCS for rapid 6D pose estimation, which is the final goal of our paper.

### 7.4.6 Weakly Supervised Learning Experiments for object detection and classification

In this first experimental section, we perform an ablation study to evaluate the impact of various components for classification and point-wise localization. We first tested our approach without
any human labeling, as a baseline. We then evaluated the gain obtained by employing various numbers of weakly labeled images for four semi-supervised strategies.

We performed these evaluations on the YCB-video dataset. This dataset contains 21 objects with available 3D models. It also has full annotations for detection and pose estimation on 113,198 training images and 20,531 test images. A subset of 2,949 test images (keyframes) is also available. Our results are reported for this more challenging subset, since most images in the bigger test set are video frames that are too similar and would report optimistic results.

For these experiments, we trained our network for 20 epochs (500 iterations per epoch) with a batch size of 4 images per domain. We used stochastic gradient descent with a learning rate of 0.001 (decay of 0.1 at epochs 10 and 16) and a Nesterov momentum of 0.9. The ResNet-50 was pre-trained on ImageNet and the weights of the first two blocks were frozen.

### 7.4.7 Unsupervised Domain Adaptation

For this experiment, we trained our model with weakly labeled synthetic images (WS) and unlabeled real images (UR). We tested three architecture configurations of domain adaptation: 1) without any domain adaptation module (WILDCAT model trained on WS), 2) with DANN (WS+UR) and 3) with MADA (WS+UR). We evaluated each of these configurations for both classification and detection. For classification, we used the accuracy metric to evaluate our model’s capacity to discriminate which objects are in the image. We used a threshold of 0.5 on classification scores to predict the presence or absence of an object. For detection, we employed the point-wise localization metric, which is a standard metric to evaluate the ability of weakly supervised networks to localize objects. For each object in the image, the maximum value in their class-specific heatmap was used to retrieve the corresponding pixel in the original image. If this pixel is located inside the bounding box of the object of interest, it is counted as a good detection. Since the class-specific heatmap is a reduced scale of the input image due to pooling, a tolerance equal to the scale factor was added to the bounding box. In our case, a location in the class-specific heatmaps corresponds to a region of 32 pixels in the original image. In Figure 7.24, we report the average scores of the last 5 epochs over 3 independent random runs for each network variation. These results a) confirm the importance of employing a domain adaptation strategy to bridge the reality gap, and b) the necessity of
Figure 7.24: Performance analysis. In (a), we compare classification accuracy and point-wise detection when no label on real images are available. In (b), we compare the performance of different training processes when different numbers of real images are weakly labeled.

<table>
<thead>
<tr>
<th>Method</th>
<th>Modality</th>
<th>Supervision</th>
<th>Full Dataset</th>
<th>AUC (ADD-S)</th>
<th>ADD-10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>PoseCNN</td>
<td>RGB</td>
<td>Pixelwise labels + 6D poses</td>
<td>Yes</td>
<td>75.9</td>
<td>24.9</td>
</tr>
<tr>
<td>PoseCNN + ICP</td>
<td>RGBD</td>
<td>Pixelwise labels + 6D poses</td>
<td>Yes</td>
<td>93.0</td>
<td>78.9</td>
</tr>
<tr>
<td>DeepHeatmaps</td>
<td>RGB</td>
<td>Pixelwise labels + 6D poses</td>
<td>Yes</td>
<td>81.1</td>
<td>25.7</td>
</tr>
<tr>
<td>FCN + ICP + ICP</td>
<td>RGBD</td>
<td>Pixelwise labels</td>
<td>Yes</td>
<td>84.0</td>
<td></td>
</tr>
<tr>
<td>FCN + StoCS</td>
<td>RGBD</td>
<td>Pixelwise labels</td>
<td>Yes</td>
<td>90.1</td>
<td></td>
</tr>
<tr>
<td>Michel et al. [42]</td>
<td>RGBD</td>
<td>Pixelwise labels + 6D poses</td>
<td>Yes</td>
<td>-</td>
<td>56.6</td>
</tr>
<tr>
<td>OURS</td>
<td>RGBD</td>
<td>Object classes</td>
<td>No (10 weakly labeled images)</td>
<td>88.7</td>
<td>68.8</td>
</tr>
<tr>
<td>OURS (multiscale inference)</td>
<td>RGBD</td>
<td>Object classes</td>
<td>Yes</td>
<td>90.2</td>
<td></td>
</tr>
<tr>
<td>OURS (multiscale inference)</td>
<td>RGBD</td>
<td>Object classes</td>
<td>No (10 weakly labeled images)</td>
<td>-</td>
<td>76.6</td>
</tr>
<tr>
<td>OURS (multiscale inference)</td>
<td>RGBD</td>
<td>Object classes</td>
<td>Yes</td>
<td>93.6</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.5: Area under the accuracy-threshold curve for 6D Pose estimation on YCB-Video dataset (ADD-S metric) and ADD metric for Occluded Linemod with threshold of 10% of the diameter (ADD-S metric for 2 objects).

having one domain discriminator $G_d^k$ for each of the X objects in the YCB database (MADA), instead of a single one (DANN). Next, we evaluate the gains obtained by employing weakly-annotated real images.

### 7.4.8 Semi-Supervised Domain Adaptation

A significant challenge for agile deployment of robots in industrial environments is that they ideally should be trained with limited annotated data, both in terms of numbers of images and of their extensiveness of labeling (no pose information, just class). We thus evaluated the performance of four different strategies as a function of the number of such weakly-labeled real images:

1. Without domain adaptation:

   (a) Real Only: Trained only on weakly labeled real images,
(b) Fine-Tuning: Trained on synthetic images and then fine-tuned on weakly labeled real images,

2. With domain adaptation:

(a) Fine-Tuning: Trained on synthetic images and then fine-tuned on weakly labeled real images,

(b) Semi-Supervised: Trained with synthetic images and weakly labeled real images simultaneously.

For 1.a and 1.b, we validate that using fine-tuning on a network pre-trained with synthetic data is preferable to training directly on real images. For 2.a and 2.b, we compare the performance of our approach trained with fine-tuning, and in a semi-supervised way (using images from both domains at the same time). We are particularly interested in comparing the two approaches 2.a and 2.b, since \[167\] achieved the lowest error rate compared to any other semi-supervised approach by only using fine-tuning.

Our results are summarized in Figure \[7.24\]. From them, we conclude that training with synthetic images improves classification accuracy drastically, especially when few labels are available. Also, our approach performs slightly better when trained in a semi-supervised setting (2.b) than with a fine-tuning approach (2.a), which is contrary to \[167\].

In this Section, we justified our architecture, as well as the training technique employed, to create a network capable of performing object identification and localisation through weak learning. In the next Section, we demonstrate how the feature maps extracted by our network can be employed to perform precise 6 DoF object pose estimation via StoCS.

### 7.4.9 6D Pose Estimation Experiments

We evaluated our full approach for 6D pose estimation on YCB-video \[48\] and Occluded-Linemod \[41\] datasets. We used the most common metrics to compare with similar methods. The average distance (ADD) metric \[36\] measures the average distance between the pairwise 3D model points transformed by the ground truth and predicted pose. For symmetric objects, the ADD-S metric measures the average distance using the closest point distance. Also, the
visible surface discrepancy [3] compares the distance maps of rendered models for estimated and ground-truth poses.

We used the same training details mentioned in section 7.4.6. Since the network architecture is fully convolutional, we also added an experiment for which we combined the output of the network for 3 different scales of the input image (at test time only).

7.4.10 YCB-Video Dataset

This dataset comprises several frames from 92 video sequences of cluttered scenes created with 21 YCB objects. The training for competing methods [48, 166, 30] is performed using 113,199 frames from 80 video sequences with semantic (pixelwise) and pose labels. For our proposed approach, we used only 10 randomly sampled weakly annotated (class labels only) real images per object class combined with synthetic images. As in [48], we report the area under the curve (AUC) of the accuracy-threshold curve, using the ADD-S metric. Results are reported in Table 7.5. Our proposed method achieves 88.67% accuracy with a limited number of weakly labeled images and up to 93.60% when using the full dataset with multiscale inference. It outperforms competing approaches, with the exception of PoseCNN+ICP, which performs similarly. However, our approach has a large computational advantage with an average runtime of 0.6 seconds per object as opposed to approximately 10 seconds per object for the modified-ICP refinement for PoseCNN. It also uses a) nearly a hundredfold less real data, and b) only using the class labels. This results thus demonstrate that we can reach fast and competitive results without the need of 6D fully-annotated real datasets.

7.4.11 Occluded Linemod Dataset

This dataset contains 1215 frames from a single video sequence with pose labels for 9 objects from the LINEMOD dataset with high level of occlusion. Competing methods are trained using the standard LINEMOD dataset, which consists in average of 1220 images per object. In our case, we used 10 real random images per object (manually labelled) on top of the generated synthetic images, using the weak (class) labels only. As reported in Table 7.5, our method achieved scores of 68.8% and 76.6% (multiscale) for the ADD evaluation metric and using a threshold of 10% of the 3D model diameter. These results compare with state-of-the-art
<table>
<thead>
<tr>
<th>Method</th>
<th>Recall Score (%)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vidal-18 [114]</td>
<td>59.3</td>
<td>4.7</td>
</tr>
<tr>
<td>Drost-10 [30]</td>
<td>55.4</td>
<td>2.3</td>
</tr>
<tr>
<td>Brachmann-16 [115]</td>
<td>52.0</td>
<td>4.4</td>
</tr>
<tr>
<td>Hodan-15 [37]</td>
<td>51.4</td>
<td>13.5</td>
</tr>
<tr>
<td>Brachmann-14 [39]</td>
<td>41.5</td>
<td>1.4</td>
</tr>
<tr>
<td>Buch-17-ppfh [116]</td>
<td>37.0</td>
<td>14.2</td>
</tr>
<tr>
<td>Kehl-16 [43]</td>
<td>33.9</td>
<td>1.8</td>
</tr>
<tr>
<td>OURS (MS)</td>
<td>55.2</td>
<td>0.6</td>
</tr>
<tr>
<td>OURS (MS) + ICP</td>
<td><strong>62.1</strong></td>
<td>6.4</td>
</tr>
</tbody>
</table>

Figure 7.25: Visual discrepancy recall scores (%) (correct pose estimation) for $\tau = 20\text{mm}$ and $\theta = 0.3$ on Occluded Linemod, based on the 6D pose estimation benchmark [3]. MS means multiscale.

methods while using less supervision and a fraction of training data. The multiscale variant (input image at 3 different resolutions) made our approach more robust to occlusions. We did not train with the full Linemod training dataset, since the dataset only has annotations for 1 object per image and our method requires the full list of objects that are in the image. Furthermore, we evaluated our approach on the 6D pose estimation benchmark [3] using the visual discrepancy metric. We evaluated our network with multiscale inference and we can see in Table 7.25 that we are among the top 3 for the recall score while being the fastest. We also tested the effect of combining ICP with StoCS. At the cost of more processing time, we obtain the best performance among the methods that were evaluated on the benchmark.

### 7.5. Safely Picking Objects in Clutter

Picking an item in the presence of other objects can be challenging as it involves occlusions and partial views. Given access to object models, one approach is to perform pose estimation for the objects and use the most likely candidate pose per object to pick the target without collisions. This approach, however, ignores the uncertainty of the perception process both regarding the target’s and the surrounding objects’ poses. This work proposes first a perception process based on state-of-the-art solutions for 6D pose estimation, which returns discrete distribution of object poses in a scene due to the nature of point cloud registration methods. Then, an open-loop, planning pipeline is proposed that follows the principles of conformant probabilistic planning...
Figure 7.26: Left: An RGB image for a scenario considered in this work. The target “Jell-o” object is located between and in close proximity to other objects. Center: Point cloud obtained from the RGB-D sensor highlighting the occlusions and partial object views. Right: Example output of pose estimation returning a discrete set of poses for objects in the scene.

(CPP). It aims to return safe and effective solutions for moving a robotic arm to perform picking in such setups. The resulting paths, given the discrete pose distribution: (a) minimize the probability of collision with the obstructing objects; and (b) maximize the probability of reaching the target item. The planning framework models the challenge as a stochastic variant of the Minimum Constraint Removal (MCR) problem. The effectiveness of the methodology is verified given both simulated and real data for two robots in different scenarios. The experiments demonstrate the importance of considering the uncertainty of the perception process in terms of safe execution. The results also show that the methodology is more effective than conservative MCR approaches, which avoid all possible object poses regardless of the reported uncertainty.

7.5.1 Generation of Discrete Pose Distributions

The first requirement is to recognize which objects are in the scene and their poses, as well as the pose of the target item. This work utilizes the perception pipeline of Fig. 7.27 for this purpose. The workspace $W$ is assumed to contain some known obstacles $O_{st}$ (e.g., a table or a shelf), and can contain any of up to $N$ objects from a set $O_{obj} = \{O_1, \ldots, O_N\}$, for which 3D models are available. There is a target item $O_t$ in the scene, for which a model is also available.

Given an input RGB image, a fully convolutional neural network (FCN) is designed to detect the objects that are present in the scene and to compute their segmentation masks. The neural network comprises a VGG16 feature encoder, followed by classification (lower branch in Fig. 2) and segmentation (top branch in Fig. 2). The classification outputs confidence scores for each object, which correspond to the probability $X_i$ of each object $O_i \in O_{obj}$ existing
in the scene. The segmentation outputs \( N \) probability masks, one corresponding to each object possibly present in the scene. Each pixel in an objects probability mask indicates the chance of the object being present at that pixel.

Given the probability maps from the segmentation, a geometric model matching process is initiated for all objects with \( X_i \) greater than a threshold (0.3 in experiments). The process samples and evaluates a large number of pose hypotheses for each object. The poses are scored based on the point cloud matching between the observed point cloud and the object model placed at hypothesized poses. The poses are then sorted based on their matching scores and clustered. The clustering iterates over the poses in order of their scores. If a pose hypothesis is close to a higher-ranked pose (within 2.5cm and 15 degrees), it is clustered with and represented by the higher-ranked one. This ensures that similar poses are not selected and the representative is the one with highest alignment score. Finally, the top \( K \) pose representatives for each object \( O_i \) are returned with scores normalized to sum up to the existence probability, \( X_i \). Denote \( p_j^i \) as the \( j \)-th pose of object \( O_i \) and \( Pr(p_j^i) \) as the probability that object \( O_i \) will appear at pose \( p_j^i \). Then:

\[
X_i = \sum_{j=1}^{K} Pr(p_j^i) \leq 1, \quad \forall i \in [1, \ldots, N].
\]  

(7.8)

The target item is assumed to be in the scene, i.e., \( X_t = 1 \). But there is uncertainty regarding its poses as well, i.e., poses \( p_j^t \) with probabilities \( Pr(p_j^t) \) are also detected for it. The number of hypotheses \( K \) for each object can vary. For simplicity, the same value is used for all objects (5

Figure 7.27: Perception pipeline developed to return the likelihood of each object existing in the scene and a discrete distribution of object poses.
7.5.2 Problem setup and notation

Path Robustness. The robustness of a path in this paper is defined based on two aspects:

1) Minimum collision probability with objects in the scene: Collisions not only cause damages but also change the scene. As a result, after every contact the scene should be re-sensed. Replanning has to be performed to produce new paths given the updated scene. Thus, a path with minimum collision probability not only minimizes damages but also reduces overall execution time to complete a task.

2) Maximum probability of reaching the target item: Since the target item $O_t$ also carries uncertainty with different pose hypotheses $p^t_j$, a situation may occur where a path with low collision risk ends up in a picking configuration with low probability. In this case, the task is incomplete and subsequent replanning has to be triggered. Therefore, maximizing the probability of reaching the target object is also important for task completion.

Definition 1. (C-space): The configuration space (C-space) $C$ of the robot arm is the set of all arm configurations. $C_p$ is defined as the set of arm configurations which end up in collision with an object pose $p$.

Definition 2. (Goal configurations): $Q_{goal}$ is a set of goal configurations from where the arm can pick the target object at poses $p^t_j$. $T = [1, \ldots, K]$ is the set of indices of all $K$ target poses. Each goal configuration $q_g \in Q_{goal}$ is associated with a set $J(q_g) \subseteq T$ indicating which target poses $q_g$ can pick. Then, the probability that a path $\pi$ from the initial configuration $q_s$ to any goal configuration $q_g$ leads to a successful picking of the target object $O_t$ is $Pr(q_g) = \sum_{j \in J(q_g)} Pr(P^t_j)$, i.e., equal to the probability that the target $O_t$ is at one of the poses $q_g$ can pick. These goal configurations are generated via an inverse kinematics (IK) solver.

Definition 3. (Survivability of a path) A path $\pi : [0, 1] \rightarrow C, \pi(0) = q_s, \pi(1) \in Q_{goal}$ survives an object $O_i$, if it does not collide with $O_i$. The survivability of a path $\pi$, denoted as $S_\pi$, is the probability that $\pi$ survives all the objects.

$S_\pi$ is computed as follows: Define $E_i$ as the event that the path $\pi$ collides with object $O_i$.
and $E_i$ as the complementary event. Then, $E_i^j$ is defined as the event that $\pi$ collides with object $O_i$ when $O_i$ is at pose $p_i^j$. Then $E_i = \bigcup_{j=1}^K E_i^j$.

The events that $O_i$ appears at different candidate poses are mutually exclusive, which indicates mutual exclusiveness of $E_i^j$. Thus, the probability that $\pi$ does not collide with $O_i$ is:

$$Pr(E_i) = 1 - Pr(E_i) = 1 - Pr(\bigcup_{j=1}^K E_i^j) = 1 - \sum_{j=1}^K Pr(E_i^j).$$ (7.9)

Then, the survivability of a path for all objects $O_i \in O_{obj}(i = 1, \ldots, N)$ is computed as:

$$S_\pi = Pr(\bigcap_{i=1}^N E_i) = \prod_{i=1}^N Pr(E_i) = \prod_{i=1}^N (1 - \sum_{j=1}^K Pr(E_i^j)).$$ (7.10)

The definition of survivability $S_\pi$ of a path provides the metric of a path’s robustness in terms of the first objective, i.e., minimum collision probability with objects. The higher $S_\pi$ is, the less risky the path is in terms of collision.

**Definition 4.** (Success probability): Define $E$ as the event that the path survives all objects and $F$ as the event that the robot arm reaches the target item. Both events $E$ and $F$ must occur for a path $\pi : q_s \rightarrow q_g$ to successfully reach the target. Define the success probability of a path to be $Succ(\pi) = Pr(E,F)$. Then, $Succ(\pi)$ can be rewritten as:

$$Succ(\pi) = Pr(E,F) = Pr(E) \cdot Pr(F) = S_\pi \cdot Pr(q_g|\pi).$$ (7.11)

Note that events $E$ (collisions) and $F$ (reach the goal) are mostly independent with one exception, which leads to defining $Pr(F)$ as the conditional probability $Pr(q_g|\pi)$. The events are not independent when the path leading to a goal $q_g$ collides with the target pose $p_t^j$ with which $q_g$ is associated. In this case, $q_g$ is no longer considered for the path as a valid goal configuration for $p_t^j$, since: (1) if the target item $O_t$ is at pose $p_t^j$, the path will be in collision with $O_t$ or (2) if the target item $O_t$ is not at pose $p_t^j$, then $q_g$ does not allow picking $O_t$ at pose $p_t^j$.

This means that $Pr(q_g)$ should be conditioned on the path. Define $\tilde{J}_\pi$ as the target poses intersected by the path. Then, the remaining valid target poses for $q_g$ should be $\tilde{J}(q_g) = \ldots$
\( J(q_g) \setminus J_\pi \) and \( Pr(q_g | \pi) \) is set to be:

\[
Pr(q_g | \pi) = \sum_{j \in J_\pi(q_g)} Pr(P_t^j).
\] (7.12)

Define the path space \( \Pi \), which includes the set of candidate paths \( \pi : q_s \rightarrow q_g \), where \( q_g \in \Omega_{\text{goal}} \). The overall objective is to find a path \( \pi^* \):

\[
\pi^* = \arg\max_{\pi \in \Pi} \text{Succ}(\pi).
\] (7.13)

### 7.5.3 Algorithmic Framework

This section describes the algorithmic approach to the problem. Consider a roadmap \( G(V, E) \), where \( V \) are vertices and \( E \) are edges. Here each vertex \( q \in V \) corresponds to an arm configuration, and each edge \( e \in E \) corresponds to the transition between two arm configurations. The roadmap \( G \) guarantees no collision between the robot and known obstacles \( O_{st} \). If an edge \( e \) connecting two configurations \( q_1, q_2 \) intersects with a pose \( P_t^j \), a label \( l_j^i \) associated with pose \( P_t^j \) is assigned to that edge and the weight for the label \( w(l_j^i) \) is equal to the appearance probability \( Pr(P_t^j) \) of the pose.

The survivability \( S_\pi \) of a path \( \pi \) depends on the probability that the objects will appear at any potential pose that the path intersects. So \( Pr(E_t^j) \) in Eq.7.10 (for event \( E_t^j \) that \( \pi \) collides with object \( O_i \) at pose \( P_j^i \)) depends on whether there is an edge \( e \) along the path \( \pi \) that has a label \( l_j^i \). If such an edge exists along the path, then \( Pr(E_t^j) = Pr(P_t^j) = w(l_j^i) \). Otherwise, \( Pr(E_t^j) = 0 \). An indicator random variable \( 1_{\pi}(j,i) \) for each pose \( j \) of each object \( i \) is defined as

\[
1_{\pi}(j,i) = \begin{cases} 
1, & \text{if } \pi \text{ carries label } l_j^i \\
0, & \text{otherwise.}
\end{cases}
\] (7.14)

Then, the survivability of a path \( S_\pi \) in a labeled roadmap can be computed as

\[
S_\pi = \prod_{i=1}^{N} (1 - \sum_{j=1}^{K} Pr(E_t^j)) = \prod_{i=1}^{N} (1 - \sum_{j=1}^{K} w(l_j^i) 1_{\pi}(j,i)).
\] (7.15)
Eq. 7.15 computes $S_p$ for a path from $q_s$ to any currently examined state $q_{curr}$ by checking the labels $L_{q_{curr}}$ the path $\pi : q_s \rightarrow q_{curr}$ carries. To compute the prospect of the path $\pi$ accurately reaching the target, two situations are considered:

(i) If $q_{curr}$ is a goal configuration $q_g$, then the probability that the path leads to the goal can be computed according to Eq. 7.12.

(ii) If $q_{curr}$ is not a goal configuration, then the current path is not a complete path yet and there is a list of indices $T_{q_{curr}} \subseteq T$ of the target poses the path $\pi : q_s \rightarrow q_{curr}$ can reach. At the beginning of a search over the roadmap, when $q_{curr} = q_s$, all goals are available $T_{q_s} = T$ (all the target poses are available). As the search proceeds, if the current path $\pi : q_s \rightarrow q_{curr}$ carries a label $l^j_t$ indicating that the path will collide with the target object $O_t$ at pose $p^j_t$, then the path and its extensions can no longer treat $p^j_t$ as a valid target pose to reach. In this case, $T_{q_{curr}}$ is updated by removing pose index $j$ from $T_{q_{curr}}$. An example is given in Fig. 7.28 (right). Then the probability for the current path $\pi : q_s \rightarrow q_{curr}$ leading to the true target will be

$$Pr(q_{curr}|\pi) = \sum_{j \in (T_{q_{curr}} \cap J_p)} Pr(p^j_t)$$

(7.16)

indicating the probability that the target $O_t$ is at one of the remaining valid target poses for the path $\pi : q_s \rightarrow q_{curr}$ and its extension to reach.

Figure 7.28: $q_{gj}$ is denoted as a goal configuration which can only grasp the target $O_t$ at pose $p^j_t$. (Left) When the path $\pi : q_s \rightarrow q_{curr}$ is found, it intersects pose $p^j_t$. This makes the goal $q_{gj}$ no longer valid for this path. The set of goals for the path will be $\{q_{g2}, q_{g3}\}$ and $T_{q_{curr}} = [2, 3]$ (Right) There are 3 candidate goals from $q_{curr}$: $q_{g1}$, $q_{g3}$, and $q_{g2}$. The goal $q_{g1}$ is not available since the path from $q_{curr}$ to $q_{g1}$ collides with pose $p^1_t$. But it is still an available goal if the path reaches $q_{g1}$ via $q_{n}$. $q_{g1}$ is treated as a valid goal but it can also be treated as an intermediate configuration to reach a potentially more promising goal $q_{g2}$. The child node for the path ending at $q_{g1}$ will be added to search twice, once as a goal node, once as a non-goal node.

With Eq. 7.12, 7.15 and 7.16, the success probability $Succ(\pi)$ of a path $\pi : q_s \rightarrow q_{curr}$ can be computed, which is used as the objective function during the search.
7.5.4 Challenge - Dynamic programming does not hold

A challenge of searching over the labeled roadmap is that the properties of dynamic programming do not hold.

Consider the setup in Fig. 7.29 (left) where the locally optimal path does not turn out to be globally optimal. This means that the search algorithm needs to keep many alternative paths that reach the same node, instead of remembering locally the optimal one. This is because the evaluation function \( \text{Succ}(\pi) \) is not an additive function of each edge’s value along a path. Instead, its computation is largely based on the set of labels the path carries (Eq. 7.15), which is a union of all sets of labels the edges carry along a path.

Figure 7.29: (Left) There are 2 poses \( p_1, p_2 \), where \( Pr(p_1) = 0.3 \) and \( Pr(p_2) = 0.4 \). There are two paths considered: the pink path \( q_s \rightarrow q_a \rightarrow q_m \) is favored vs. the blue one \( q_s \rightarrow q_b \rightarrow q_m \) locally as it has a lower collision probability. Nevertheless, both paths go through pose \( p_2 \) afterwards (\( q_m \rightarrow q_g \)). Thus, the optimal path to \( q_g \) is actually the blue one, which only collides with pose \( p_2 \) with probability 0.4. (Right) There are 3 poses: \( p_1 \) for object \( O_1 \) with \( Pr(p_1) = 0.3 \) and \( Pr(p_2) = 0.3 \), while \( p_2 \) belongs to \( O_2 \) with \( Pr(p_2) = 0.4 \). Again the pink path \( q_s \rightarrow q_a \rightarrow q_m \) is locally favored since it has higher survivability \( 1 - Pr(p_1) = 0.7 \) than that of the blue path \( q_s \rightarrow q_b \rightarrow q_m \) \( (1 - Pr(p_2) = 0.6) \). But when both paths reach \( q_g \), the pink path has survivability \( 1 - Pr(p_1) - Pr(p_2) = 0.4 \), which is lower than that of the blue path \( (1 - Pr(p_2))(1 - Pr(p_2)) = 0.42 \).

While omitted due to space limitations, a reduction of the considered problem to the computationally hard MCR problem is straightforward. For the MCR problem, it has been argued that greedy search (where only the best local path is stored at each node) still guarantees an optimal solution if the optimal path encounters each obstacle once [168]. Even with this assumption for the problem considered here, greedy search still has a chance to fail to find the optimum (Fig. 7.29 (right)).

7.5.5 MaxSuccess Search

As mentioned above, greedy search is not guaranteed to be optimal. Complete search methods, such as exact search, are considered here. (The greedy version is also implemented for
The proposed complete method follows the exact search for the MCR problem but adapts it to address the survivability objective defined here, which includes probabilities for the object poses and uncertainty regarding the target. The algorithm stores a path to a node only if the label set of the path is not a superset of that of any path found reaching the same node. It theoretically guarantees optimality since a path with a superset of labels cannot have a higher success probability $\text{Succ}(\pi)$ than that of a path reaching the same node with a subset of labels.

### Algorithm 11: MaxSuccess Exact Search

**Input:** $G(V, E), q_s, Q_{goal}, T = [1, \ldots, K]$

**Output:** $\pi^*$

1. $Q \leftarrow \text{ADD}(q_s, L_{q_s} = \emptyset, T_{q_s} = T, \mathbb{1}_{q_s} = false)$
2. while goal not found do
   3. $q_{curr} \leftarrow Q.top()$
   4. if $\mathbb{1}_{q_{curr}} = true$ then
      5. return $\pi : q_s \rightarrow q_{curr}$
   6. for each $q_{neigh} \in \text{Adj}(G, q_{curr})$ do
      7. $L_{q_{neigh}} = L_{q_{curr}} \cup L_e(q_{curr}, q_{neigh})$
      8. if not ISSUPERSET($L_{q_{neigh}}$) then
         9. $q_{neigh, \pi} \leftarrow q_{curr, \pi} \cup e(q_{neigh}, q_{curr})$
         10. $S_{q_{neigh, \pi}} \leftarrow \text{GETSURVIVAL}(L_{q_{neigh}})$
         11. $T_{q_{neigh}} \leftarrow \text{UPDATEGOALS}(L_{q_{neigh}})$
         12. $Pr(q_{neigh | \pi}) \leftarrow \text{GETREACH}(T_{q_{neigh}})$
         13. $\text{Succ}(q_{neigh, \pi}) \leftarrow S_{q_{neigh, \pi}} \cdot Pr(q_{neigh | \pi})$
         14. $Q \leftarrow \text{ADD}(q_{neigh, L_{q_{neigh}}, T_{q_{neigh}}, false})$
      15. if $q_{neigh} \in Q_{goal}$ then
         16. if ISVALID($q_{neigh, T_{q_{neigh}}}$) then
            17. $Q \leftarrow \text{ADD}(q_{neigh, L_{q_{neigh}}, T_{q_{neigh}}, true})$

The method is outlined in Alg. It receives as input the roadmap $G(V, E)$, the start arm configuration $q_s$, a set of goal configurations $Q_{goal}$ and a list of all target poses indices $T$. A priority queue $Q$ (line 1) prioritizes nodes with higher $\text{Succ}(\pi : q_s \rightarrow q)$. Higher $S_\pi$ is the primary tie breaker and smaller path cost the secondary. Each state $q$ is specified with a label set $L_q$ indicating the labels the path $\pi : q_s \rightarrow q$ carries and the set of indices of the target poses $T_q \subseteq T$ the path $\pi$ can reach. An indicator $\mathbb{1}_q$ is assigned to a node to indicate whether it is a goal. The node with the highest $\text{Succ}()$ value is expanded as $q_{curr}$ (line 3). If it is a goal, then the goal with the highest success value $\text{Succ}()$ is found (line 4-5). If not, the algorithm considers all adjacent nodes $q_{neigh}$ of $q_{curr}$ and computes the labels that the path from $q_s$ to $q_{neigh}$ via $q_{curr}$
If it is a non-goal node, it will only be added once (line 14). A goal search terminates when a goal node and whether the path then added to $Q$ as a non-goal node ($L_q = false$) (line 14). Then the algorithm checks if it is a goal node and whether the path $\pi: q_s \rightarrow q_{\text{curr}} \rightarrow q_{\text{neigh}}$ still treats this goal $q_{\text{neigh}}$ as a valid one (line 15-16). If $q_{\text{curr}}$ meets both conditions, it is added to $Q$ as a goal ($L_q = true$) (line 17). The search terminates when a goal $q_g \in Q_{\text{goal}}$ has been found (line 5) or there is no solution.

A key observation is that each node has a chance to be added to $Q$ twice in a single iteration (line 14 and line 17). If it is a non-goal node, it will only be added once (line 14). A goal carries with a union operation (line 6-7). If the set of labels the path carries is not a superset of any label set of previously stored paths $\pi: q_s \rightarrow q_{\text{neigh}}$ (line 8), the path $\pi: q_s \rightarrow q_{\text{curr}} \rightarrow q_{\text{neigh}}$ is stored; its corresponding survivability and the indices of available target poses the path can reach are computed (line 9-11). The probability for the path to reach the target is computed in line 12 using Eq. 7.16 and $\text{Succ}(\cdot)$ value of the path can be computed (line 13). The node is then added to $Q$ as a non-goal node ($L_q = false$) (line 14). Then the algorithm checks if it is a goal node and whether the path $\pi: q_s \rightarrow q_{\text{curr}} \rightarrow q_{\text{neigh}}$ still treats this goal $q_{\text{neigh}}$ as a valid one (line 15-16). If $q_{\text{curr}}$ meets both conditions, it is added to $Q$ as a goal ($L_q = true$) (line 17). The search terminates when a goal $q_g \in Q_{\text{goal}}$ has been found (line 5) or there is no solution.

A key observation is that each node has a chance to be added to $Q$ twice in a single iteration (line 14 and line 17). If it is a non-goal node, it will only be added once (line 14). A goal

<table>
<thead>
<tr>
<th>Table 1 (narrow passage)</th>
<th>Table 2 (clutter)</th>
<th>Shelf 1 (narrow passage)</th>
<th>Shelf 2 (clutter)</th>
</tr>
</thead>
</table>

Table 7.30: Results on 4 benchmarks (two for tabletop and two for a shelf) evaluating the number of objects collided during execution for (top row) different number of pose hypotheses under uncertainty level 4 and (middle row) different uncertainty levels for 4 poses per object. Each column corresponds to the benchmark shown below it. The target object is the baseball in the tabletop cases and the water bottle in the shelf cases. Each data point corresponds to the mean of 35 different potential sensing/roadmap inputs per ground truth condition.
node can be treated as a goal configuration to grasp an object, but it can also be treated as an intermediate node to reach a more promising goal configuration. For that reason, the node is added as a non-goal node (line 14) as usual. Then, the node is only added as a goal node (line 17) if it is still valid (Fig. 7.28(right)).

7.5.6 Experimental Setup

The proposed planning framework in 7.5.3 is evaluated on (A) simulated sensing data and (B) data from a real-world setup. The following alternatives are also considered:

1. an Optimistic Shortest Path (OSP) planner - which ignores the presence of the movable objects $O_{obj}$,

2. an MCR search (Exact and Greedy) - which aims to minimize the number of collisions with all poses, and

3. an MCR Most Likely Candidate (MCR – MLC) search, which considers only the most likely pose for each object and aims to minimize the number of collisions.

Both the greedy and exact versions of MCR and MaxSuccess have been implemented. For better reading of Fig. 7.30, the exact version for both methods is used.

The methods are evaluated on an Intel Xeon E5-1660 processor, where the simulations model a 7-DoF Kuka LBR iiwa14 and a 7-DoF Yaskawa Motoman SDA10F robot manipulator. This paper uses a suction-based gripper at the end of the robot arm.

7.5.7 Simulation Experiments

The experiment is first performed on simulated sensing data since it makes larger-scale experiments possible and provides an intuition on how the algorithms work in response to different levels of uncertainty. Four benchmarks are created to simulate robotics applications (Fig. 7.30 bottom). Benchmarks 1-2 are tabletop scenarios, where the arm needs to reach the target object so as to pick it from above. Benchmarks 3-4 are highly-constrained shelf scenarios with side picking. The target object is either placed in a narrow passage or in clutter.
The ground truth pose of each object is defined first for each scene. Pose hypotheses are generated according to probability distributions centered at the ground truth. The number of poses per object is sampled between 1 to 7. Different levels of uncertainty are defined. At level 1, poses are generated with $\pm 0.5$ cm for translation error and $\pm 5$ degrees orientation error. At level 7, poses have $\pm 3.5$ cm and $\pm 35$ degrees noise. The sensing noise in intermediate levels (2-6) is linearly interpolated between level 1 and 7. The probability for each pose is assigned based on its distance from the ground truth (the closer, the higher the probability).

The algorithms are evaluated according to (1) the number of objects colliding in the ground truth scene and (2) success rate in reaching the true target.

Once a planning scene is defined, a roadmap is also generated for the arm using a Probabilistic Roadmap ($\text{PRM}$) algorithm \cite{169}. The connectivity of the roadmap is defined similar to the ($\text{PRM}^*$) variant, which achieves asymptotic optimality \cite{132}, i.e., each node is connected to at least $k^* = k_n \cdot \log(n) = e(1 + 1/d)\log(n)$ neighbors, where $e$ is the base of the natural logarithm, $d$ the dimension of the search space ($d=7$) and $n$ the number of samples. The size of each roadmap constructed for the arm is $n = 5000$. 35 roadmaps have been generated for each ground truth. 6,860 total different sensing inputs have been generated. Fig. 7.30 provides the number of objects collided during path execution. The top row reports the collisions for different number of pose hypotheses under uncertainty level 4. The middle row reports statistics for different number of pose hypotheses under uncertainty level 4. The middle row reports statistics for different uncertainty levels for 4 poses per object.

In the tabletop benchmarks, the OSP paths collide with 1.54 objects on average and MCR – MLC with 0.74. In contrast, both MCR and MSE work well with much fewer collisions. As the uncertainty increases, the number of collisions for MCR methods start to increase, while the MSE algorithm remains almost down to zero collisions. Overall, MCR and MSE return result in few collisions. Nevertheless, Fig. 7.31 indicates that the success rate for MCR methods is not as
Figure 7.32: Experimental results on real vision system in 3 scenarios (1) target in objects clutter (top row) (2) target in narrow passage (middle row) (3) target in front of objects arch (bottom row). The second and third column demonstrate the number of objects collided during path execution and the success rate of reaching the target, respectively, as the evaluation metrics for 6 methods (including both exact and greedy version of MaxSuccess and MCR).

The success rates for MCR and MSE methods are tabulated in Table 7.1 and Table 7.2, respectively, with the corresponding success rates being 68.4% and 56.2%. The high success rate is expected since the uncertainty of the target pose is low. As a result, MCR may avoid collisions but does not lead to the true target pose. Since the MSE methods take both safety and goal reachability to form the success function \( \text{Succ}(\pi) \), the corresponding success rate is high on the tabletop scenes (99.5%).

The shelf benchmarks are even more challenging. The fact that the object must be reached from the side in a limited space increases the collision risk. Despite that, MCR and MSE remain highly safe (0.34 and 0.22 collisions, respectively). Again, the MCR method is conservative in terms of collisions at the expense of failing to reach the true target pose. In Fig. 7.31, the success rate for MCR drops to (42.6%) in benchmark 4 (clutter, shelf), while MSE is still able to succeed 83.6% of the trials.

### 7.5.8 Real-world Experiments

After verifying the effectiveness of the proposed MaxSuccess algorithm with simulated sensing data, an evaluation with real data took place. Fig. 7.33 shows the setup used. An Azure Kinect camera is mounted on top of a humanoid Motoman SDA10F robot to enable an overhead view...
of the objects on the table. The experiment focuses on overhand picks in a tabletop setup. 33 images have been taken from the camera with diverse scenarios:

1) **target in objects clutter** - The target is surrounded by multiple objects. The robot has to reason about the uncertainty of the objects to reach the target without collision.

2) **target in narrow passage** - The target is placed between 2 or 3 tall objects, which creates a narrow passage. The arm has to reach deep to pick the target.

3) **target in front of objects arch** - An arch is created by three objects, where one of the objects is put on top of the other two objects. The target is placed a little bit in front of the arch. Since the robot is reaching the target with overhand picks, the relative location between the arch and the target has to be carefully examined to succeed.

10 YCB objects are selected in the experiment. The ”pudding box”, ”gelatin box” and ”meat can” are selected as the target objects in different scenes. All the images have been processed in the perception pipeline described in Section 7.5.1. An object is treated as present if the probability prediction is over 0.3. For each object detected in an image, $K = 5$ object poses have been returned with their corresponding probabilities as the outcome of pose estimation and pose clustering. All the data regarding the 6D pose hypotheses can be found online for further reference.

Then, the proposed planning framework takes these pose hypotheses as input. The process generates 5 roadmaps for each planning scene to evaluate the quality of paths returned by each method. The same metrics are used (1) number of objects in collision; and (2) success rate of reaching the target.

Fig. 7.32 demonstrates the performance of different methods on the real vision system. As the scenarios have no obvious risk-free picking path, OSP suffers from a high number of collisions. $MCR - ML$ also has a relatively high number of collisions, which confirms the idea that the distribution of object poses have to be considered instead of only trusting the most likely pose. The true pose for an object may not be the top response of pose estimation. In every scenario, the methods $MSE$ and $MSG$ outperforms $MCR - G$ and $MCR - E$, which claim to be good at finding safe paths. $MCR - G$ and $MCR - E$ have low success rate of reaching the target as
Figure 7.34: Left image shows the grasping configuration chosen by MCR method. It avoids any risk of colliding with the arch but has a large chance to miss the target. The right one shows the grasping configuration chosen by MaxSuccess. It reasons about the uncertainty of the sugar box and the target object pudding box together and comes up with a better solution which can reach the target accurately with low risk of collisions.

<table>
<thead>
<tr>
<th></th>
<th>OSP</th>
<th>MCR-G</th>
<th>MCR-E</th>
<th>MLC</th>
<th>MSG</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>time(s)</td>
<td>0.030</td>
<td>0.031</td>
<td>0.033</td>
<td>0.031</td>
<td>0.045</td>
<td>0.046</td>
</tr>
</tbody>
</table>

Table 7.6: Path cost and planning time.

these methods tend to find conservative paths to avoid obstacles, sacrificing the reasoning about target poses. This observation can be particularly made from the arch scenario (Fig. 7.32, the most bottom-right bar graph). Fig. 7.34 illustrates the advantage of the proposed method over MCR in the arch scenario.

The path cost (Euclidean distance between arm configurations along the path) and the planning time have also been computed for reference, though they are not the main objectives considered in this paper.
Chapter 8

Task-driven Perception and Manipulation for Constrained Placement of Unknown Objects

Object placement in tight spaces is a challenging problem in robot manipulation. In contrast to a simpler pick-and-drop problem, only specific object poses will allow it to fit in a tight space. Such scenarios occur in logistics applications, such as packing items into boxes, or in service robotics, such as inserting a book into a gap in a bookshelf. Recent work has focused on variants of this problem, such as bin-packing [79, 10] and table-top placement in clutter [80]. Nevertheless, in many cases a geometric and textured 3D model for the manipulated object is assumed to be known. Possessing such high-fidelity models is expensive both in terms of time and effort. In several setups it becomes infeasible to build models due to the wide variety of objects to be manipulated and the resources required for obtaining the models. Some recent robot manipulation systems [81, 82] have shown the capacity of picking novel and previously unseen objects from clutter. These systems, however, typically assume no constraints for the object’s placement. Therefore, the object is grasped with any feasible and stable grasp without reasoning about placement. Some alternatives do not require exact models of objects but operate with category-level prior information. Examples include an approach based on sparse keypoint representations [87] and deep reinforcement learning [84]. While the employed representations can guide manipulation planning solutions, they do not account for safety as they do not consider geometric or physical constraints.

This work targets pick-and-place problems where the task imposes constraints on the placement pose. The capabilities of a manipulator impose limitations on what placement poses are reachable depending on the grasp, making certain grasps more desirable than others. This requires careful reasoning to select the pick that will allow the desired placement. This will be referred to as the pick-and-constrained-placement problem. In the context of this problem,
Figure 8.1: A demonstration of pick-handoff-place returned by the proposed framework for inserting a previously unknown object in a constrained space. This work does not focus on the last step of precise, closed-loop insertion but how to reason about the object’s shape so as to safely bring it to the opening of the placement area. Only a subset of placement poses allow the object to fit into the target area. The experiments performed use a larger margin than what is shown here (2 cm instead of 1 cm shown).

it is possible that a feasible placement pose is not directly attainable using a pick-and-place operation. Instead, it may require a re-grasping of the object or a hand-off to be executed.

Solutions to such problems typically need object models for collision checking, which this work does not assume. Picking, placement, and re-grasping actions need to be computed given partial viewpoints of the object acquired from the sensor, as in Fig. 8.2. This work approaches the pick-and-constrained-placement problem without prior object models as integrated perception and manipulation planning. The objective is to place the entire object safely inside the desired goal region without any collisions, while minimizing the time and sensing operations required to complete the task.

One option is to pick the object with a task-agnostic grasp that solely interacts with the visible part of the object as in pick-and-drop systems [81, 82]. After picking, the object’s shape can be completely reconstructed by manipulating it to different configurations in front of the sensor. Then, geometric planning can be performed based on the reconstructed model. Nevertheless, not only will this option be very time-consuming given the object must be moved
Figure 8.2: (Left) Figure shows a partial view of the object from the sensor. A conservative estimate of the object considers both the observed and the unobserved parts of the object and knowledge of the support surface. An optimistic estimate considers only the observed parts. (Right) The estimates are updated as the object is manipulated to different sensor viewpoints.

to multiple viewpoints, but using a goal-agnostic pick may not allow the constrained placement without multiple regrasps. A key point of this work is that constrained placement task can be successfully completed without a complete object model. This motivates a dynamic estimate of the object’s shape to solve the problem, and a planning approach capable of using and updating such a representation on the fly.

The algorithmic solution proposed simultaneously operates over conservative and optimistic estimates of the object’s 3D volume, as in Fig. 8.2. The conservative estimate considers the entire volume attached to the object, which has not been observed by the sensor as part of the object. The optimistic estimate considers only the object’s observed region to be its complete representation. While the conservative estimate ensures that the manipulation is safe, the optimistic estimate guides the action selection when no solution can be found for the conservative estimate. Both estimates are dynamically updated by incorporating new viewpoints, which are selected such that a safe-to-execute constrained placement solution can be found with minimal sensing.

To efficiently obtain these dynamic estimates, this work proposes to utilize a simple volumetric representation. Similar to occupancy-grids [170] often used in the context of robot navigation to store the occupied and free space, this representation stores whether a voxel in
the object’s reference frame is occupied, unoccupied or unobserved. Instead of utilizing fixed-size grids or octrees to store the volumetric information, the representation maintains sets of occupied and unobserved voxels. This minimalistic representation provides efficiency at the cost of building exact models but proves to be sufficient to solve the considered problem.

The effectiveness of the proposed approach is demonstrated by developing a robotic system that picks a previously unseen object from a table-top and places it in a constrained space as in Fig. 8.1. The system comprises of a dual-arm manipulator with a single RGB-D sensor, a vacuum-based end-effector and an adaptive, finger-based hand. Additionally, the system features handoffs for transferring objects between the two arms and a strategy to adjust the computed motion trajectories during real-world execution given sensing updates. Handoff is a re-grasping strategy that allows more flexibility in solving constrained placement problems. Closed-loop execution handles stochastic in-hand motions of objects resulting from unmodeled physical forces like gravity, inertia and grasping contacts.

240 real-world manipulation experiments are performed to compare the proposed solution and the straightforward pick-sense-and-place alternative. The experiments demonstrate that the proposed pipeline is both robust and efficient in handling objects with no prior models within the limitations of the end-effector and the sensor. It achieves a success rate of 95.82%, which is much higher than an alternative that commits to a pick without manipulation planning and performs object reconstruction from heuristic viewpoints without utilizing the conservative volumetric representation. The proposed pipeline results in fewer sensing operations and achieves faster execution times.

8.1. Problem Setup and Notation

This section formulates the integrated perception and manipulation planning problem for constrained placement.

Object representation: A rigid object can be defined by a region occupied by the object $O^* \subseteq \mathbb{R}^3$ in its local reference frame that represents its shape. Given a pose $P \in SE(3)$, the

\footnote{Videos and supplementary algorithmic description can be found at: \url{https://robotics.cs.rutgers.edu/task-driven-perception/}}
Figure 8.3: (left) The proposed framework considers as input RGB-D images and the target object mask and builds a shape representation based on the observed and the occluded part of the object. (center) It simultaneously operates over a conservative and an optimistic estimate of the object’s volume to compute a sequence of manipulation and sensing actions for pick-and-constrained-placement. The object is dynamically updated during the manipulation until a safe to execute sequence of action is available. (right) Online adaption is performed, which is informed by pose-tracking to counter the effect of stochastic in-hand motion of the object, which is not modeled during planning.

region occupied by the object at $P$ is denoted by $O_p^*$. It should be noted that a geometric model is not available for the object to be manipulated, i.e., $O^*$ is unknown. Thus, $O$ defines an object representation over which manipulation planning can operate. In general, $O \neq O^*$. $O$ is derived from an initial view of the object given point cloud and image segmentation. The resulting object model is typically incomplete, and may not be sufficient to safely place the object in a constrained area.

Constraining placement: Given an object at an initial pose $p_{\text{init}} \in SE(3)$, the goal of the constrained placement problem is to transfer $O^*$ to a pose $p_{\text{target}} \in SE(3)$, such that $O^*_{\text{target}} \subset R_{\text{place}}$ where $R_{\text{place}} \subset \mathbb{R}^3$ is the target placement region.

Manipulation Planning: Manipulation planning for constrained placement involves computing a sequence of manipulation actions (picks, placements, re-grasps) that can move the object $O^*$ from $p_{\text{init}}$ to $p_{\text{target}}$, which successfully solves a constrained placement task. Such a solution consists of motions of the arms denoted by $\Pi$ parameterized by the time of the motions. $\Pi(0)$ is the initial arm configuration, and $\Pi(1)$ has an arm placing the object at $p_{\text{target}}$.

Integrated Perception and Manipulation Planning: Given that the true object geometry $O^*$ is unknown and planning can make use only of the partial object representation $O$, perception actions are also necessary. These actions can update the object representation $O$ by manipulating it to desirable configurations in front of the sensor and obtaining additional sensing
information. Thus, the problem involves computing a sequence of perception and manipulation actions, such that: i) the object after executing the sequence of actions ends up inside the defined constraints, i.e., $O_{final}$ is within $R_{place}$, where $P_{final}$ is the resultant pose of the object after applying the actions; ii) and the returned sequence of perception and manipulation actions minimizes the task execution time.

8.2. Proposed pipeline

This section presents the proposed pipeline as shown in Figure. 8.3. Given as input RGB-D image of the scene and the target object mask $T_{mask}$, the object representation $O$ is initialized with it’s origin at the centroid of the 3D point cloud segment corresponding to $T_{mask}$ and the reference frame at identity rotation with respect to the camera frame. Within a voxel grid centered at the origin, each voxel is labeled as either 1) observed and occupied $S$, 2) unobserved $U$, or 3) observed and unoccupied, i.e., empty voxels that are implicitly modeled as a set of points \( \{ p \in \mathbb{R}^3 \mid p \notin S \cup U, \| p - \text{origin}(O) \| < D_{\text{max}} \} \), for a maximum dimension parameter $D_{\text{max}} = 30cm$. The representation is stored as a point-set $O$ that consists of two mutually exclusive sets of points $S$ and $U$ in $\mathbb{R}^3$. $S$ is a set of seen points on the surface of the object that are observed by the RGB-D sensor. $U$ is a set of unseen points in space that have not been observed by the sensor given its observations but have a non-zero probability of belonging to the target object. Thus $O = S \cup U$, where, $S \cap U = \phi$.

A set of grasps and placements are computed simultaneously over a conservative estimate and an optimistic estimate of the object’s volume. The conservative estimate corresponds to $O$, while the optimistic estimate only considers the observed part of the object, i.e., $S$. Manipulation planning is performed considering the grasps and placements computed over $O$. The objective is to compute a sequence of these manipulation actions and corresponding arm motions, which allow to connect a grasp to a placement pose. Any manipulation planning solution computed over $O$ can be directly executed in the real-world as it is necessarily collision-free with respect to the true object shape $O^*$, given that $O^* \subseteq O$. Often no solution can be found for the task as $O$ may significantly overestimate $O^*$. In such a scenario, the object is picked and manipulated to acquire new observations, thereby updating $O$. 
The choice of picking point is critical as it might influence the solution once $O$ has been updated. For this reason, manipulation planning is performed over the optimistic estimate of the object’s volume. In this case, all actions after picking, such as re-grasps and placements are computed over $S$. If no placements are achievable given $S$, the problem is not solvable, since $S \subseteq O^\ast$. If a solution is found for $S$, it informs the selection of the picking point over $O$.

The next decision is the selection of the next best view. It is selected among a set of pre-defined discrete viewpoints with an objective of exposing the highest number of unobserved voxels in $U$. This is found by rendering $S$ at each of the viewpoints and computing the number of voxels in $U$ that are visible, given the rendered image. The selected viewpoint is most-likely to reduce the conservative volume of the object. The object is then moved to this viewpoint and $O$ is updated.

The size of the set $O$ (and thus the conservative volume) is largest at initialization. Any update to $O$ either removes a point $p \in U$ (if it is observed to be empty) or $p$ can be moved from $U$ to $S$. To update $O$, the observed segment $s'$ at time $t$ is transformed to the object’s local frame based on the estimated pose $P_t$. For each point $p$ on the transformed point cloud, its nearest neighbor $p^S \in S$ and $p^U \in U$ are found. If $|p^S - p| < \delta_c$ where $\delta_c$ is the correspondence threshold, $p$ is considered to be already present. Otherwise, if $|p^U - p| < \delta_c$, $p^U$ is removed from $U$ and added to $S$. Finally, the method iterates over all points in $U_P$ to remove points in $U$, which belong to the empty part of space based on the currently observed depth image. Applying these constraints in the update significantly reduces the drift that occurs in simultaneous updates to the object’s pose and shape.

Grasps and placements are re-computed over the updated object estimate and manipulation planning is performed again. This process is repeated until either a solution is found for the constrained placement task or the algorithm runs out of a maximum number of trials. This means that the pipeline does not require the object to be completely reconstructed, but only enough to compute a safe-to-execute solution for the placement task.

8.3. System Design & Implementation
Fig. 8.4 shows the hardware setup. It comprises a dual-arm manipulator (Yaskawa Motoman) with two 7-dof arms. The left arm is fitted with a narrow, cylindrical end-effector with a vacuum gripper; and the right arm is fitted with a Robotiq 2-fingered gripper. A single RGB-D sensor (Kinect Azure) is mounted on the robot overlooking both the picking and the placement regions. The sensor is configured in Wide-FOV mode to capture images at 720p resolution with a frequency of up to 20Hz. Below are the implementation details corresponding to different components of the proposed pipeline for this hardware setup.

**Grasp computation:** Grasp sets $G_l$ and $G_r$ are computed over the object representation $O$ by ensuring stable geometric interaction with the observed part of the object $S$ and being collision-free with both $S$ and $U$, thereby ensuring safe and successful execution. It is also crucial for the success of manipulation planning to have large, diverse grasp sets at its disposal. This is distinct from the typical objective of grasp generation modules that primarily focus on the quality of the top (few) returned grasps. For instance, in Fig. 8.3 the grasps are spread out over $O$ with different approach directions, which provide options to manipulation planning and aid solution discovery.

Vacuum grasps $G_l$ are computed by uniformly sampling pick points and their surface normals from $S$, and ranked in quality by their distance from the shape centroid. The grasp set $G_r$ for the fingered gripper samples a large set of grasps over $O$ according to prior work [83]. Sampled grasps are pushed forward along the grasp approach direction until the fingers collide with points from $S$ or $U$, and ranked by the alignment between the finger and contact region on $S$.

**Placement Computation:** Given the placement region $R_{\text{place}}$, and the object representation $O$, two boxes are computed, 1) the maximum volume box $B_{\text{place}}$ within $R_{\text{place}}$ and 2) the minimal volume box $B_O$ that encloses $O$. Candidate placement poses correspond to configurations of $B_O$ which fit within $B_{\text{place}}$. A discrete set of configurations ($= 24$) for the box is computed by placing $B_O$ at the center of $B_{\text{place}}$ and validating all axis-aligned rotations. Any pose in the returned set $P_{\text{place}}$ is a candidate $P_{\text{target}}$.
**Manipulation Planning:** The input to manipulation planning is the estimated object representation $O$, the grasp sets available for both arms, $\mathcal{G}_l, \mathcal{G}_r$, and the placement poses $\mathcal{P}_{\text{place}}$. Manipulation planning returns a sequence of prehensile manipulation actions that ensure a collision free movement (II) of the arms and $O$ such that the object is transferred from $p_{\text{init}}$ to some $p_{\text{target}} \in \mathcal{P}_{\text{place}}$. In the absence of any errors, the execution of these actions solves the constrained placement task.

As a part of the task planning framework, a probabilistic roadmap [169] consisting of 5000 nodes is constructed using the PRM$^*$ algorithm [132], for each of the arms. The grasps and placements for each arm can be attained by corresponding grasping, and placement configurations of the arms, obtained using Inverse Kinematics solvers. Beginning with the initial configuration of the arms, the high-level task planning problem becomes a search over a sequence of the manipulation actions, achievable by the pick, place or handoff configurations. This is described in the form of a forward search tree [171] which operates over the same roadmap [172] by invalidating edges (motions) that collide with the object, or the other arm. The search tree is further focused by only expanding pick-place and pick-handoff-place action sequences. Each such sequence can be achieved through a combination of different choices of grasping, handoff, and placement configurations. The search traverses the set of options for grasps in the descending order or quality, and returns the first discovered solution that successfully achieves a valid target placement ($p_{\text{target}} \in \mathcal{P}_{\text{place}}$).

**Shape and Pose Tracking:** The object pose $P'$ changes over time with the gripper manipulating it, where $E^t \in SE(3)$ denotes the gripper pose at time $t$. Between consecutive timestamps for a perfect prehensile manipulation, $\Delta P^{t-1:t} = \Delta E^{t-1:t}$ which is the change in the gripper’s pose. Tracking is introduced to account for non-prehensile within-hand motions which violates this nicety.

The object segment at any time $s^t$ is computed from a) points lying in a pre-defined region of interest in the reference frame of the gripper, and b) by eliminating the points corresponding to the gripper’s known model. Object pose update $\Delta P^{t-1:t}$ is computed in three steps:

1) Assuming rigid attachment of the object with the end-effector, the transformation, $\Delta E^{t-1:t}$ is applied to the object segment in previous frame $s^{t-1}$ to obtain the expected object segment at time $t$, $s^t$. 

2) To account for any within hand motion of the object, a transformation is computed between $s'^t$ and the observation $s^t$ via ICP. While $\Delta P^{t-1} = \Delta E^{t-1} \cdot \Delta P_{ICP}$ provides a good estimate of relative pose between consecutive frames, accumulating such transforms over time can cause drift.

3) A final point-set registration process is utilized to locally refine the pose. An ICP registration step with a strict correspondence threshold is performed between the object representation $(O)$ at pose $P^t = P^{t-1} \cdot \Delta P^{t-1}$, and the current observation $s^t$. The resulting transformation is applied to $\Delta P^{t-1}$, and correspondingly $P^t$.

During manipulation, when a new viewpoint is encountered, the output of pose tracking is utilized to update the object’s shape which assists tracking in future frames.

**Reaction to Sensing Updates:** Given a manipulation planning solution $\Pi$, the objective is to ensure that any errors in execution or non-prehensile grasping interactions are addressed. At any point in time $t$, $\Pi(t)$ describes how the arms are configured. Assuming prehensile grasps, the expected object pose $P^*_t$ can be estimated. Tracking returns the current estimate $P^t$. If $P^t \neq P^*_t$ the remainder of the motion has to be adjusted to account for $\Delta P = P^*_t - P^t$. Large $\Delta P$ errors may require complete re-planning of $\Pi$. In this work these adjustments are performed before handoffs, and placements by locally adapting $\Pi$. 

Figure 8.5: Objects used in the experiments (top). Examples of initial configurations (left). Examples of placement constraints (right).
8.4. Experimental Setup

This section describes the setup for the experiments performed to measure the efficacy of the proposed pipeline and the developed system in solving the *pick-and-constrained-placement* problem. Given the dual-arm manipulator, objects are placed on a table-top in front of the left arm (vacuum gripper), with the target placement region centrally aligned in front of the robot, reachable by both arms. The constrained placement solutions can therefore involve a direct placement by the left arm, or a handoff-placement with the right arm. Following describe the different parameters of the setup followed by the evaluation metrics.

*Objects:* Experiments are performed over 5 YCB\(^73\) objects (Fig. 8.5) of different shapes and sizes. It should be noted that *no models* are made available to the method.

*Initial Configuration:* For each stable resting pose of the object in front of the left arm, rotations were uniformly sampled along the axis perpendicular to the plane of the table. Different initial configurations of the object will affect the nature of the task planning solution by virtue of a) different available initial picks, and b) different conservative shape representation based on how much of the object is unseen at the configuration. Configurations with limited reachable grasps are ignored. The height of the table is known in advance and it is used to obtain the initial point cloud segment for the object.

*Placement Region:* An opening is created on the table surface where the object needs to be placed. This corresponds to the placement task. Two placement scenarios are evaluated as shown in Fig. 8.5 (bottom right). Using the measures of three canonical dimensions measured from the object, the first class of opening size allows four out of six approach directions for placement to fit, while the other only allows two approach directions. An error tolerance of 2.00\(cm\) is considered in the dimension of the opening. The idea is that more constraints (lesser approach directions) need deliberate planning to choose precise grasp and handoff sequences that allow the placement. Evaluating the insertion with a lower margin would need to consider the sensor accuracy, errors in detection of the target placement region and the accuracy of an insertion controller, which are not the focus of this work.

*Evaluation metrics:* Given the task, the pipeline is responsible to pick the object and reconfigure it such that it ends up within the desired placement region on top of the table. A
simple control strategy is used to insert the object into the hole and measure the success of the task. It utilizes cartesian control to incrementally lower the object until the joint-limits are reached or a collision is observed. The object is then dropped. Success (S) denotes the percentage of trials that result in collision-free, successful insertion of objects within the constrained opening, while Marginal Success (MS) records trials where the object grazes the boundaries of the constrained space during a successful insertion. In terms of quality metrics, Task planning time records open-loop manipulation planning, Move time records the time the robot is in motion, and Sensing actions counts the number of times the robot actively re-configures the object to acquire sensor data from a new viewpoint.

Baseline - Complete Shape Reconstruction: The baseline (shown in Fig. 8.5) picks the object with a task-agnostic pick (i.e., any pick that works) and reconstructs the entire object by moving to pre-defined viewpoints. Manipulation planning is performed on the reconstructed shape to find and execute a solution for constrained placement.

A drawback of this approach is that committing to a task-agnostic pick might preclude solutions, which might have been possible with a different pick. For instance, the initial pick might not allow a direct placement or in some cases even obstruct handoffs. Another drawback is that the amount of object reconstruction required depends on the task. It can be inefficient to fully reconstruct the object if a robust solution with partial information can be found. Finally, even with a large number of perception actions, some parts of the objects might be missing, which can still lead to execution failures. For instance, this can happen if say the bottom surface is not reconstructed and fingered grasps interact with the unmodeled part of the object during execution.

8.5. Results

240 trials are performed with combinations of object sets, initial configurations and placement constraints. Out of these, 120 experiments use the Baseline pipeline shown in Fig. 8.5 and the remaining 120 use the proposed pipeline. The results for Baseline (BL) and Baseline + Handoff (HO) are derived from the same set of physical experiments. Fig. 8.7 shows the outcome of the experiments. The failures include Placement failures where the final act of placement fails to insert the object, Handoff failures where executing the transfer of object
Figure 8.6: Qualitative results indicating different solution modes of the proposed pipeline.

Table 8.1: Evaluating the task success rate of the proposed manipulation pipeline against a baseline. Overall 240 manipulation trials were executed, where the results corresponding to Baseline and Baseline + handoff are derived from the first set and the results for the proposed pipeline are derived from the second set. S indicates successful insertion in the constrained space, and MS stands for marginal success, where the object made contact with the boundary of the constrained space but the task still succeeded.

between the arms fails, and No Solution cases when planning fails and nothing is executed.

**Baseline (BL):** The baseline corresponds to the shape reconstruction pipeline but without the option for handoffs. Once picked with a task-agnostic grasp, the object is moved in front of the sensor at a predefined pose, and RGB-D images are captured from 4 different viewpoints by rotating the object along the global Z-axis by an angle of \( \pi/2 \). Views are merged to obtain the object’s reconstruction. Manipulation planning is then invoked to find a pick-and-placement (no handoff) solution with the left arm if it exists. The baseline achieves a very low success rate (Table 8.1) and the most dominant failure mode is No Solution (Fig. 8.7) since the initially chosen grasp might not allow task completion. This implies that given the selected grasp, a reachable, collision-free placement configuration cannot be found for the arm.
Baseline + Handoff (HO): An improvement over BL, this allows the manipulator an additional option of transferring the object to the fingered gripper which can then be used to reorient and place it in the constrained space. The overall success rate increases significantly when additional handoff actions are available. Nonetheless, the handoff by itself can be seen as a constrained placement problem, and as this approach commits to a pick for object reconstruction without manipulation planning, it could still lead to No solution cases specially for relatively smaller sized objects such as for the Mustard bottle (Fig. 8.7). The grasps with the fingered gripper are computed assuming that the reconstructed geometry is indeed the complete model of the object. However, views across a single rotation are not sufficient to complete the object shape. Unlike the proposed approach, the baseline does not consider the unseen part of the object as a collision geometry. This causes grasps to collide with the unmodeled parts of the object during execution (Handoff failures). The baseline approach performs re-sensing after it picks the object. The re-sensing action prevents any inconsistency due to in-hand motion of the object during the pick. Nevertheless, any in-hand motion that occurs after the reconstruction
does not get accounted for and can result in Placement failures. Placement can also fail if the reconstructed geometry is an under-approximation of the true object geometry.

**Proposed Pipeline:** The proposed pipeline discovers four classes of solutions (Fig 8.6) that compose a sequence of picks, updates, handoffs and placements. The key benefit is that it chooses the mode of operation based on the problem at hand, and tries to (a) perform the minimum number of sensing actions (b) with a minimum number of manipulation actions (c) in a robust fashion that accounts for non-prehensile errors (d) while guaranteeing safe execution and successful task completion. The results reflect that it achieves all of the above by leveraging the object representation, integrated perception and planning in the pipeline, and closed loop execution to achieve a success rate of 95.82%.

The proposed pipeline eliminates the cases of No Solution by performing manipulation planning with a large, diverse, and robust set of grasps. It ensures successful execution of the task by conservative modeling of the unseen parts of the object to avoid collision and by tracking the shape representation to account for any in-hand motion of the object and adjusting the computed plan. The failure cases for this approach are due to failures in tracking. If the within-hand motion is too drastic, motion plans might not be found for local adjustments to the initially computed solution.

As indicated in Fig. 8.6 and Table. 8.2, the proposed solution can find one of the four solution modes with varying solution quality. The advantage in terms of efficiency comes from the fact that the proposed solution requires additional sensing in only 38% of the runs and the mean number of sensing actions is 1.36 as opposed to the 4 additional sensing actions in every run for the baseline approach. Additionally, the object representation allows task planning with multiple grasping options even before picking thereby increasing the number of single-shot pick-and-place solutions with less motion time. The overall execution time reduces significantly due to the combination of these factors.

**Demonstrations and Publicly-shared Data:** On top of the benchmark, additional demonstrations show the capability of the proposed system. The first demonstration is performed over mugs, some with and some without handles, with the handles being occluded in the first viewpoint. Such a case imposes ambiguity for shape completion approaches, but is solved with the proposed pipeline as demonstrated in the accompanying video.
The second demonstration presents the task of flipping objects and placing them on the table. Without models, object placement tasks can either be specified relative to constraints in the environment or relative to the initial pose. Following data items corresponding to all the manipulation runs for the proposed solution are made publicly available at 1) Task specification: Initial RGB-D data, object segment, placement region. 2) RGB-D data at 20Hz for the executed trajectory. 3) Robot arm transformations and grasping status for both grippers. 4) Relative pose estimates returned by the tracking module for every frame. The data can be used as a manipulation benchmark or to study tracking shapes and poses of objects in-hand during manipulation.

8.6. Judging the Intent of Pointing Actions with Robotic Arms

Collaborative robotics requires effective communication between a robot and a human partner. This work proposes a set of interpretive principles for how a robotic arm can use pointing actions to communicate task information to people by extending existing models from the related literature. These principles are evaluated through studies where English-speaking human subjects view animations of simulated robots instructing pick-and-place tasks. The evaluation distinguishes two classes of pointing actions that arise in pick-and-place tasks: referential pointing (identifying objects) and locating pointing (identifying locations). The study indicates that human subjects show greater flexibility in interpreting the intent of referential pointing compared to locating pointing, which needs to be more deliberate. The results also demonstrate the effects of variation in the environment and task context on the interpretation of pointing. Our corpus, experiments and design principles advance models of context, common sense reasoning and communication in embodied communication.

8.6.1 Communicating Pick-and-Place

This section provides a formalization of pick-and-place tasks and identifies information required to specify them.

Figure 8.8: Demonstrations of the proposed pipeline’s operation (left) in the presence of shape ambiguity (right) on the object flipping task.
Figure 8.9: A pick-and-place task requires a referential pointing action to the object (orange cube) at the initial position, and a locating pointing action to a final placement position (dotted cube). Such an action by a robot (in red) can also be accompanied by verbal cues like “Put that there.”

**Manipulator:** Robots that can physically interact with their surroundings are called manipulators, of which robotic arms are the prime example.

**Workspace:** The manipulator operates in a 3D workspace $\mathcal{W} \subseteq \mathbb{R}^3$. The workspace also contains a stable surface of interest defined by a plane $S \subseteq \mathcal{W}$ along with various objects. To represent 3D coordinates of workspace positions, we use $x \in \mathcal{W}$.

**End-effector:** The tool-tips or end-effectors are geometries, often attached at the end of a robotic arm, that can interact with objects in the environment. These form a manipulator’s chief mode of picking and placing objects of interest and range from articulated fingers to suction cups. A subset of the workspace that the robot can reach with its end-effector is called the reachable workspace. The end-effector in this work is used as a pointing indicator.

**Pick-and-place:** Given a target object in the workspace, a pick-and-place task requires the object to be picked up from its initial position and orientation, and placed at a final position and orientation. When a manipulator executes this task in its reachable workspace, it uses its end-effector. The rest of this work ignores the effect of the object’s orientation by considering objects with sufficient symmetry. Given this simplification, the pick-and-place task can be viewed as a transition from an initial position $x_{\text{init}} \in \mathcal{W}$ to a final placement position $x_{\text{final}} \in \mathcal{W}$. Thus, a pick-and-place task can be specified with a tuple

$$PAP = (o, x_{\text{init}}, x_{\text{final}}).$$

**Pointing Action:** Within its reachable workspace the end-effector of the manipulator can attain
different orientations to fully specify a reachable pose \( p \), which describes its position and orientation. The robots we study have a directional tooltip that viewers naturally see as projecting a ray \( r \) along its axis outward into the scene. In understanding pointing as communication, the key question is the relationship between the ray \( r \) and the spatial values \( x_{\text{init}} \) and \( x_{\text{final}} \) that define the pick-and-place task.

To make this concrete, we distinguish between the target of pointing and the intent of pointing. Given the ray \( r \) coming out of the end-effector geometry, we define the target of the pointing as the intersection of this ray on the stable surface,

\[ x^* = r \cap S. \]

Meanwhile, the intent of pointing specifies one component of a pick-and-place task. There are two cases:

- **Referential Pointing:** The pointing action is intended to identify a target object \( o \) to be picked up. This object is the referent of such an action. We can find \( x_{\text{init}} \), based on the present position of \( o \).

- **Locating Pointing:** The pointing action is intended to identify the location in the workspace where the object needs to be placed, i.e., \( x_{\text{final}} \).

We study effective ways to express intent for a pick-and-place task. In other words, what is the relationship between a pointing ray \( r \) and the location \( x_{\text{init}} \) or \( x_{\text{final}} \) that it is intended to identify? To assess these relationships, we ask human observers to view animations expressing
pick-and-place tasks and classify their interpretations. To understand the factors involved, we investigate a range of experimental conditions.

8.6.2 Experiment Setup

Each animation shows a simulated robot producing two pointing gestures to specify a pick-and-place task. Following the animation, viewers are asked whether a specific image represents a possible result of the specified task.

Robotic Platforms The experiments were performed on two different robotic geometries, based on a Rethink Baxter, and a Kuka IIWA14. The Baxter is a dual-arm manipulator with two arms mounted on either side of a static torso. The experiments only move the right arm of the Baxter. The Kuka consists of a single arm that is vertically mounted, i.e., points upward at the base. In the experiments the robots are shown with a singly fingered tool-tip, where the pointing ray is modeled as the direction of this tool-tip.

Workspace Setup Objects are placed in front of the manipulators. In certain trials a table is placed in front of the robot as well, and the objects rest in stable configurations on top of the table. A pick-and-place task is provided specified in terms of the positions of one of the objects.

Objects The objects used in the study include small household items like mugs, saucers and boxes (cuboids), that are all placed in front of the robots.

Motion Generation The end-effector of the manipulator is instructed to move to pre-specified waypoints, designed for the possibility of effective communication, that typically lie between the base of the manipulator and the object itself. Such waypoints fully specify both the position and orientation of the end-effector to satisfy pointing actions. The motions are performed by solving Inverse Kinematics for the end-effector geometry and moving the manipulator along these waypoints using a robotic motion planning library [173]. The motions were replayed on the model of the robot, and rendered in Blender.

Pointing Action Generation Potential pointing targets are placed using a cone $C(r, \theta)$, where $r$ represents the pointing ray and $\theta$ represents the vertex angle of the cone. As illustrated in Fig 8.10, the cone allows us to assess the possible divergence between the pointing ray and the actual location of potential target objects on the rest surface $S$.

Given a pointing ray $r$, we assess the resolution of the pointing gesture by sampling $N$
Referential pointing
Locating pointing
Imprecise, judged correct
Imprecise, judged incorrect
Precise, judged correct
Precise, judged correct

"Put that ..."
"... there"

Figure 8.11: The image shows the differences between referential (top) and locating pointing (bottom), demonstrated on a robotic manipulator, Kuka IIWA14. An overlay of the object is shown at the placement location where locating pointing needs to be directed. Human subjects are more flexible in the interpretation of referential pointing than with that of locating pointing.

Object poses $p_i, i = 1 : N$ in $P = C(r, \theta) \cap S$—the intersection of the pointing cone with the rest surface. While $p_i$ is the 6d pose for the object with translation $t \in R^3$ and orientation $R \in SO(3)$ only 2 degrees-of-freedom $(x, y)$ corresponding to $t$ are varied in the experiments. By fixing the $z$ coordinate for translation and restricting the $z$-axis of rotation to be perpendicular to $S$, it is ensured that the object rests in a physically stable configuration on the table.

The $N$ object poses are sampled by fitting an ellipse within $P$ and dividing the ellipse into 4 quadrants $q_1 \ldots q_4$ (See Figure 8.10 (C)). Within each quadrant $q_i$ the $N/4 (x, y)$ positions are sampled uniformly at random. For certain experiments additional samples are generated with an objective to increase coverage of samples within the ellipse by utilizing a dispersion measure.

Speech Some experiments also included verbal cues with phrases like ‘Put that there’ along with the pointing actions. It was very important for the pointing actions and these verbal cues to be in synchronization. To fulfill this we generate the voice using Amazon Polly with text written in SSML format and make sure that peak of the gesture (the moment a gesture comes to a stop) is in alignment with the peak of each audio phrase in the accompanying speech. During the generation of the video itself we took note of the peak moments of the gestures and then manipulated the duration between peaks of the audio using SSML to match them with gesture peaks after analyzing the audio with the open-source tool PRAAT (www.praat.org).
8.6.3 Data Collection

Data collection was performed in Amazon Mechanical Turk. All subjects agreed to a consent form and were compensated at an estimated rate of USD 20 an hour. The subject-pool was restricted to non-colorblind US citizens. Subjects are presented a rendered video of the simulation where the robot performs one referential pointing action, and one locating pointing action which amounts to it pointing to an object, and then to a final location. During these executions synchronized speech is included in some of the trials to provide verbal cues.

Then on the same page, subjects see the image that shows the result of the pointing action. They are asked whether the result is (a) correct, (b) incorrect, or (c) ambiguous.

To test our hypothesis, we studied the interpretation of the two pointing behaviors in different contexts. Assuming our conjecture and a significance level of 0.05, a sample of 28 people in each condition is enough to detect our effect with a 95% power. Participants are asked to report judgments on the interpretation of the pointing action in each class. Each participant undertakes two trials from each class. The range of different cases are described below. Overall, the data collection in this study involved over 7,290 responses to robot pointing actions.

8.6.4 Experimental Conditions

We used our experiment setup to generate videos and images from the simulation for a range of different conditions.

---

3The data, code, and videos are available at https://github.com/malihealikhani/That_and_There.
**Referential vs Locating**

In this condition, to reduce the chances of possible ambiguities, we place only one mug is on the table. The Baxter robot points its right arm to the mug and then points to its final position, accompanied by a synchronized verbal cue, “Put that there.”

We keep the motion identical across all the trials in this method. We introduce a variability in the initial position of the mug by sampling 8 random positions within conic sections subtending 45°, 67.5°, and 90° on the surface of the table. New videos are generated for each such position of the mug. This way we can measure how flexible subjects are to the variation of the initial location of the referent object.

To test the effect for the locating pointing action, we test similarly sampled positions around the final pointed location, and display these realizations of the mug as the result images to subjects, while the initial position of the mug is kept perfectly situated.

A red cube that is in the gesture space of the robot, and is about twice as big as the mug is placed on the other side of the table as a visual guide for the subjects to see how objects can be placed on the table. We remove the tablet that is attached to Baxter’s head for our experiments.

**Effect of speech** In order to test the effect of speech on the disparity between the kinds of pointing actions, a set of experiments were designed under the Referential vs Locating method with and without any speech. All subsequent methods will include verbal cues during their action execution. These cues are audible in the video.

**Reverse Task**

One set of experiments are run for the pick-and-place task with the initial and final positions of the object flipped during the reverse task. As opposed to the first set of experiments, the robot now begins by pointing to an object in the middle of the table, and then to an area areas towards the table’s edge, i.e., the pick and place positions of the object are ‘reversed’.

The trials are meant to measure the sensitivity of the subjects in pick trials to the direction of the pointing gestures and to the absolute locations that the subjects thought the robot was pointing at.

This condition is designed to be identical to the basic Referential vs Locating study, except
A cluttered trial consists of collecting the response from a human subject when the position of the referential pointing action lies between two objects. The motions are still executed on the Baxter’s right arm.

**Different Robotic Arm**

In order to ensure that the results obtained in this study are not dependent on the choice of the robotic platform or its visual appearance, a second robot—a singly armed industrial Kuka manipulator—is also evaluated in a Referential vs Locating study (shown in Figure 8.11).

**Cluttered Scene**

To study how the presence of other objects would change the behavior of referential pointing, we examine the interpretation of the pointing actions when there is more than one mug on the table. Given the instructions to the subjects, both objects are candidate targets. This experiment allows the investigation of the effect of a distractor object in the scene on referential pointing.

We start with a setup where there are two mugs placed on the table (similar to the setup in Figure 8.13). One is a target mug placed at position $x_{object}$ and a distractor mug at position $x_{distractor}$. With the robot performing an initial pointing action to a position $x_{init}$ on the table. Both the objects are sampled around $x_{init}$ along the diametric line of the conic section arising from increasing cone angles of $45^\circ$, $67.5^\circ$, and $90^\circ$, where the separation of $x_{object}$, and $x_{distractor}$ is equal to the length of the diameter of the conic section, $D$. The objects are then positioned on the diametric line with a random offset between $[-D/2, D/2]$ around $x_{init}$ and along the line. This means that the objects are at various distances apart, and depending upon the offset, one of the objects is nearer to the pointing action. The setup induces that the nearer mug serves as the object, and the farther one serves as the distractor. The motions are performed on the Baxter’s...
right arm. The camera perspective in simulation is set to be facing into the pointing direction. The subjects in this trial are shown images of the instant of the referential pointing action.

**Natural vs Unnatural scene**

In this condition we study how the contextual and physical understanding of the world impacts the interpretation of pointing gestures. We generate a scenario for locating pointing in which the right arm of the *Baxter* points to a final placement position for the cuboidal object on top of a stack of cuboidal objects but towards the edge which makes it physically unstable. The final configurations of the object (Figure 8.14) shown to the users were a) object lying on top of the stack b) object in the unstable configuration towards the edge of the stack and c) object at the bottom of the stack towards one side. New videos are generated for each scenario along with verbal cues.

The pointing action, as well as the objects of interest, stay the identical between the natural, and unnatural trials. The difference lies in other objects in the scene that could defy gravity and float in the unnatural trials. The subjects were given a text-based instruction at the beginning of an unnatural trial saying they were seeing a scene where “gravity does not exist.”

![Figure 8.14: The diagram shows the three different configurations of the placement of a blue cuboid object evaluated in the Natural vs Unnatural trials.](image)

**Different verbs**

To test if the effect is specific to the verb *put*, we designed a control condition where everything remained the same as the Referential vs Locating trials except the verb *put* which we replaced with *place, move* and *push*. Here again we collect 30 data points for each sampled $x^*$. 
Figure 8.15: The aggregated results from the referential versus spatial trials for the Baxter and Kuka robots. The locations of the responses correspond to the center of the circles, and are plotted in the coordinate frame centered at the position of the pointing action, marked with \( \times \). The circles show the fraction of correct (grey), incorrect (black) and ambiguous (white) responses.

8.6.5 Analysis

Referential vs Locating

We study how varying the target of the pointing action from a referent object to a part of the space changes the interpretation of the pointing action by comparing the interpretation of the position of the pointing action \( x^* \) in each condition.

Figure 8.15 shows the results of the experiment. The plot shows the spread of correct, incorrect, ambiguous responses over the sampled positions about the location of referential vs locating pointing actions. The referential data demonstrates the robustness of the interpretation. Most of the responses were overwhelmingly correct, for both robots, in interpreting a referent object in the pick part of a pick-and-place task. The locating pointing shows a much higher sensitivity to an accuracy of \( x^* \) with respect to the true final placement. This comes up as a
larger incidence of incorrect and ambiguous responses from the human subjects. This trend is true for the reverse trial as well.

While the study attempts to separate out and measure the critical aspects of the interpretation of robotic pointing actions some ambiguities like those arising out of perspective of the camera being projected onto a simulated 2D video or image are unavoidable. We suspect that the observed stretch of correct responses in spatial trials is due to perspective.

To test our hypothesis that Referential pointing is interpreted less precisely than Locating pointing we performed a Chi-squared test and compared the proportion of correct, incorrect and ambiguous responses in referential and spatial trials. The results of the test shows that these two classes are statistically significantly different ($\chi^2 = 13.89, p = 0.00096$).

To study if we are observing the same effects in the results of the reverse trial, no speech trial and the Kuka trial, we ran an equivalence test following the two one-sided tests method as described in [174], where each test is a pooled z-test with no continuity correction with a significance level of 0.05. We found changing the robot, removing the speech and changing the direction of the pointing action to make no difference in the interpretation of locating pointing and referential pointing within any margin that is less than 5%.

**Natural vs Unnatural**

As shown in Table 8.3 we observed in the natural scene, when the end-effector points towards the edge of the cube that is on top of the stack, subjects place the new cube on top of the stack or on the table instead of the edge of the cube. However, in the unnatural scene, when we explain to subjects that there is no gravity, a majority agree with the final image that has the cube on the edge. To test if this difference is statistically significant, we use the Fisher exact test [175].
Figure 8.16: The scatter plot represents the spread of responses where human subjects chose the nearer cup (green), farther cup (red), and ambiguous (white). The x-axis represents the absolute difference between the distances of each cup to the locations of pointing, the y-axis represents the total distance between the two cups.

The test statistic value is 0.0478. The result is significant at $p < 0.05$.

**Different verbs**

The results of the Chi-squared test shows that in spatial trials when we replace *put* with *place*, *push* and *move*, the differences of the distributions of *correct*, *incorrect* and *ambiguous* responses are not statistically significant ($\chi = 0.2344$, $p = 0.971$). The coefficients of the multinomial logistic regression model and the $p$-values also suggest that the differences in judgments with different verbs are not statically significant ($b < 0.0001$, $p > 0.98$).

**Cluttered**

The data from these trials show how human subjects select between the two candidate target objects on the table. Since the instructions do not serve to disambiguate the target mug, the collected data show what the observers deemed as the *correct* target. Figure 8.16 visualizes subjects’ responses across trials. The location of each pie uses the x-axis to show how much closer one candidate object is to the pointing target than the other, and uses the y-axis to show the overall imprecision of pointing. Each pie in Figure 8.16 shows the fraction of responses across trials that recorded the nearer (green) mug as correct compared to the farther mug (red). The white shaded fractions of the pies show the fraction of responses where subjects found the gesture ambiguous.
As we can see in Figure 8.16, once the two objects are roughly equidistant the cups from the center of pointing (within about 10cm), subjects tend to regard the pointing gesture as ambiguous, but as this distance increases, subjects are increasingly likely to prefer the closer target. In all cases, wherever subjects have a preference for one object over the other, they subjects picked the mug that was the nearer target of the pointing action more often than the further one.

8.6.6 Human Evaluation of Instructions

After designing and conducting our experiments, we became concerned that subjects might regard imprecise referential pointing as understandable but unnatural. If they did, their judgments might combine ordinary interpretive reasoning with additional effort, self-consciousness or repair. We therefore added a separate evaluation to assess how natural the generated pointing actions and instructions are. We recruited 480 subjects from Mechanical Turk using the same protocol described in our Data Collection procedure, and asked them to rank how natural they regarded the instruction on a scale of 0 to 5.

The examples were randomly sampled from the videos of the referential pointing trials that we showed to subjects for both the Baxter and Kuka robots. These examples were selected in a way that we obtained equal number of samples from each cone. The average rating for samples from the 45º, 67.5º and 90º cone are 3.625, 3.521 and 3.650 respectively. For Kuka, the average rating for samples from the 45º, 67.5º and 90º cone are 3.450, 3.375, and 3.400. Overall, the average for Baxter is 3.600, and for Kuka is 3.408. The differences between Kuka and Baxter and the differences across cones are not statistically significant ($t \leq |1.07|, p > 0.1$). Thus we have no evidence that subjects regard imprecise pointing as problematic.

8.6.7 Design Principles

The results of the experiments suggest that locating pointing is interpreted rather precisely, where referential pointing is interpreted relatively flexibly. This naturally aligns with the possibility for alternative interpretations. For spatial reference, any location is a potential target. By contrast, for referential pointing, it suffices to distinguish the target object from its distractors.

We can characterize this interpretive process in formal terms by drawing on observations
from the philosophical and computational literature on vagueness. Any pointing gesture starts from a set of candidate interpretations $D \subseteq W$ determined by the context and the communicative goal. In unconstrained situations, locating pointing allows a full set of candidates $D = W$. If factors like common-sense physics impose task constraints, that translates to restrictions on feasible targets $CS$, leading to a more restricted set of candidates $D = CS \cap W$. Finally, for referential pointing, the potential targets are located at $x_1 \ldots x_N \in S$, and $D = \{x_1 \ldots x_N\}$.

Based on the communicative setting, we know that the pointing gesture, like any vague referring expression, must select at least one of the possible interpretations. We can find the best interpretation by its distance to the target $x^*$ of the pointing gesture. Using $d(x, x^*)$ to denote this distance, gives us a threshold

$$\theta = \min_{x \in D} d(x, x^*).$$

Vague descriptions can’t be sensitive to fine distinctions. So if a referent at $\theta$ is close enough to the pointing target, then another at $\theta + \epsilon$ must be close enough as well, for any value of $\epsilon$ that is not significant in the conversational context. Our results suggest that viewers regard 10cm (in the scale of the model simulation) as an approximate threshold for a significant difference in our experiments.

In all, we predict that a pointing gesture is interpreted as referring to $\{x \in D | d(x, x^*) \leq \theta + \epsilon\}$. We explain the different interpretations through the different choice of $D$.

**Locating Pointing**

For unconstrained locating pointing, $x^* \in D$, so $\theta = 0$. That means, the intended placement cannot differ significantly from the pointing target. Taking into account common sense, we allow for small divergence that connects the pointing, for example, to the closest stable placement.

**Referential Pointing**

For referential pointing, candidates play a much stronger role. A pointing gesture always has the closest object to the pointing target as a possible referent. However, ambiguities arise when
the geometries of more than one object intersect with the $\theta + \varepsilon$-neighborhood of $x^\ast$. We can think of that, intuitively, in terms of the effects of $\theta$ and $\varepsilon$. Alternative referents give rise to ambiguity not only when they are too close to the target location ($\theta$) but even when they are simply not significantly further away from the target location ($\varepsilon$).
Chapter 9
Discussion

The research presented in this thesis began in the year 2016, inspired by the group’s participation in Amazon picking challenge (APC). APC was a global robot manipulation competition where the task for the manipulator was to pick a specified list of items from cluttered bins in a shelf. A superset of these objects were provided to the participating teams ahead of time. The challenge on the perception side was to detect the target objects in the scene and compute grasps over them given RGB-D sensor data. By this time, convolutional neural networks (CNNs) were shown to be very successful in the task of object detection and image segmentation for common semantic categories when trained on large-scale labeled datasets. Nevertheless, it was not clear how this success would translate to training over a large variety of products in the logistics domain and for harder tasks such as pose estimation. One of the challenges was that it was infeasible to manually acquire training data by placing combinations of objects in all possible configurations and to annotate such data. The annotation task is even more involved for 6D pose estimation.

A more scalable solution to this problem is to use synthetic data for training. The data generation and training techniques that existed at the time could not effectively utilize synthetic data to solve object recognition due to the domain gap problem. Initial work presented in this thesis demonstrated the advantage of using physics simulators in the data generation process for object recognition. Chapter 3 shows that utilizing the constraints of the environment and generating physically-realistic data can train object detectors that perform better in the real world as compared to standard data augmentation techniques. It was also shown that the trained detectors can self improve over time by automatically collecting and labeling real images. The physical constraints essentially align the distribution of object poses between the data generated in simulation and the real world images taken in that environment. Chapter 6 exploits this
alignment in pose distribution via adversarial training (with unlabeled real images) to learn object instance segmentation and pose quality evaluation function. The learned networks, without any manual annotation, show robust performance in challenging scenarios with multiple object instances and dense clutter.

Learning in simulation scales well when dealing with relatively structured environments where object models are known in advance. There are several scenarios where the object models are not known in advance or scenarios where it might not be scalable to acquire models for each object instance. For such cases, Chapter 8 presents an object representation and a manipulation planning framework that operates over previously unseen objects. Not assuming a category-level shape prior or known geometric models and operating directly over the sensor data makes this manipulation pipeline safe to execute and scalable.

Beyond the scalability aspect, the thesis focuses on developing robust algorithms that combines the learning in simulation with online optimization. Chapter 4 presents an object pose estimation algorithm called Stochastic Congruent Set Matching (StoCS) that computes an SE(3) transformation between an object model and its probabilistic segmentation in an observed point clouds. It exploits the geometric information from the known object model to sample over the probabilistic segmentation representation and search for the best transformation. The algorithm was shown to be robust in object pose estimation when only synthetic data was used for training (Chapter 4), when weak labels are used for training (Chapter 7) and when a heuristically defined function was utilized to assign probabilities for in-hand object (Chapter 7). A Monte-Carlo Tree Search (MCTS) approach (Chapter 5) and an Integer Linear Programming (ILP) solution (Chapter 6) are considered and evaluated for scene-level reasoning with physical constraints. The objective is to deal with cluttered scenarios where the object’s geometry may not be sufficient to distinguish the true object pose among the hypothesis set. Interactions between objects and with the environment provide constraints for the selection of true pose given the noise from sensor, segmentation or the ambiguities due to occlusion. The MTCS formulation can consider complex physical constraints but the ILP-based solution results in a computationally efficient approach and scales well to a clutter with large number of object instances.
Overall, the thesis presents techniques with varying level of relevance across different application scenarios. Different problems in robot manipulation have different sensing requirements. The constraints of these commonly encountered problems in robot manipulation (as shown in Figure 9.1) can be mapped to some of the proposed techniques. For problems that involve task-agnostic picking such as the pick-and-drop task, it is often not required to reason about the pose of the object being picked. The task involves picking any object from a pile with any feasible grasping configuration. The perception task is to segment the object in clutter. Simulating representative piles via physics simulation (Chapter 3) and learning visibility boundaries (Chapter 6) are very relevant tools in such scenarios. Reasoning about the pose and shape of objects become more critical in case the objects are picked for the purpose of some specific placement task such as hanging the mug or inserting object in a tight space. Techniques developed in Chapter 4 for model-based pose estimation and in Chapter 8 for model-free pose estimation are directly applicable for such tasks. In some cases, as shown in Figure 9.1, a higher level task planning might require reasoning about the dependencies between objects to complete the desired goal such as picking a specific target object placed under other objects. In such scenarios scene-level reasoning presented in Chapter 5 and Chapter 6 would be critical. Reasoning at the scene-level also helps resolve ambiguities in pose estimation due to occlusions. Finally, estimating the uncertainty is a key component in many of these problems. It is crucial to maintain an uncertainty due to the ambiguities arising from a) severe occlusion in some viewpoints for example in-hand objects or b) when the test data distribution is different from training distribution, for example when training with synthetic data. Chapter 4 provides a way to sample and score pose hypotheses based on probabilistic segmentation. This
notion of uncertainty can be updated based on a more extensive scene-level reasoning or can be directly used in the manipulation context as shown in Chapter 7.

9.1. Future Work

The past few years has seen a significant progress in model-based pose estimation. Recent approaches focus on learning to directly predict 6D pose [48, 49] from images in an end-to-end manner. While efficient, these approaches lose out in terms of explainability and modeling the uncertainty in case the data distribution changes. On the other hand, the work presented in this thesis generates a large set of pose hypotheses by sampling over learned representations and performs geometric and physics-based validation on top of it. This proves to be more adaptive and robust to the changes in data distribution. A limitation of this approach, however, is the computational cost of generating and validating a large number of pose candidates. A future work, for the MCTS-based scene-level reasoning could explore learning policies or heuristics for the combinatorial search process as more data is acquired from the process. This would allow more exploration when there is high uncertainty and later exploit the experience for fast prediction. Another topic of research, following up from the work on learning to assign pose quality scores in Chapter 6 could involve building more explainable learned components for model-based pose estimation. For example developing a learned model that could sample poses based on pointset matching or a model that could perform physics-based validation on sampled poses. There has been some recent work that explore similar topics [180, 181].

While there has been a rapid growth in instance-level pose estimation technology, it is unclear how such techniques would generalize to unseen object instances. One of the ways this problem is currently studied is by categorizing objects into semantic categories and representing them either with a canonical frame [85] or via a sparse set of semantic keypoints [87]. While the sparse keypoint representation is good for task specification and can avoid problems due to large variance in shapes within object categories, such a representation is not sufficient for geometric planning. A future work could look into combining the sparse keypoint based object representation with the sensor-based planning discussed in Chapter 8. It might also be interesting to assign separate object categories for geometric planning that are based on
primitive shapes.

Finally there is category-agnostic reasoning where no prior is assumed for the object’s category or shape. Some of the open questions in this domain include a) segmenting previously unseen objects, [182] [183] b) shape completion based on symmetry or past experience of seeing a similar object, c) finding relative transformation between different viewpoints of objects. While there are similar problems being addressed in the computer vision community, the problem in robotics are different because objects are smaller, with varying levels of geometric and visual features, and are often subjected to large occlusions. Other differences involve the presence of additional modalities such as depth data, tactile sensing and the availability of manipulation skills for active recognition.
References


Alexander Krull, Frank Michel, Eric Brachmann, Stefan Gumhold, Stephan Ihrke, and Carsten Rother. 6-dof model based tracking via object coordinate regression. In ACCV 2014.


