Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Learning 6D Object Pose from Point Clouds

Dissertation
with the aim of achieving a doctoral degree at the
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
of Universität Hamburg

submitted by
Ge Gao
Matrikelnr.: 6991266
Hamburg, 2021

1^{st} Supervisor:
Prof. Dr. Simone Frintrop
Department of Informatics,
Universität Hamburg, Germany

2^{nd} Supervisor:
Prof. Dr. Jianwei Zhang
Department of Informatics,
Universität Hamburg, Germany

Abstract

The goal of 6D object pose estimation is to find the 3D rotation and 3D translation between a known object coordinate frame and a coordinate reference frame. It is an important research topic in the field of computer vision, because knowing the 6D pose of an object is an important prerequisite for many robotic applications such as grasping and dexterous manipulation. The majority of existing methods mainly rely on color as the primary data modality. They either regard the depth information as auxiliary information or completely ignore it. However, depth information contains geometric information regarding the object surface, and it should be a promising data modality for this topic. This thesis investigates what depth information can contribute to the 6D object pose estimation problem. We focus on developing learning-based approaches for 6D object pose estimation using only depth information.

The first approach we present is CloudPose. CloudPose is a deep learning-based system that regresses to 6D object poses from depth information represented by point clouds. We use point clouds as the input and geometry-based pose refinement. CloudPose uses separate deep networks for rotation and translation regression. We argue that the axis-angle representation is a suitable rotation representation for deep learning, and use a geodesic loss function for rotation regression. Ablation studies show that these design choices outperform alternatives such as the quaternion representation and L2 loss, or regressing translation and rotation with the same network. Although the structure of CloudPose is simple, experimental results have shown competitive results on public datasets.

One drawback of CloudPose is that it is not robust against noisy input data. To overcome this drawback, another system called CloudAAE is further proposed. CloudAAE is an autoencoder-based 6D object pose regressor especially robust against the depth input data's noise. The autoencoder learns a latent vector that encodes the 6D object pose data and is invariant to noise in the input data. Then the latent code is used as the input to two separate networks for rotation and translation regression. Comparing with CloudPose, CloudAAE has a better performance on datasets with noisy depth data.

Both CloudPose and CloudAAE are supervised learning-based systems and require a large amount of annotated training data. It is often desired to train on synthetic data because the cost of manually annotating 6D object poses on real data is very high. However, it is often expensive to synthesize RGB images because of the large visual reality gap between synthetic and real RGB images. In contrast, this visual reality gap is considerably smaller and easier to fill for depth information. Hence, we present a lightweight data synthesis pipeline called CloudSyn that creates synthetic point cloud segments for training. CloudSyn only requires texture-less 3D object models and the desired viewpoints, and it is cheap in terms of both computation time and the hardware storage. Our data synthesis process is three orders of magnitude faster than the commonly applied approaches rendering RGB image data.

When evaluating CloudPose and CloudAAE, we noticed that a clean segment without background noise often results in a more accurate 6D pose estimate during our experiments. In other words, the quality of the object segmentation has an impact on the system performance. To this end, we present a point cloud over-segmentation framework named SSV, which is short for saliency-guided adaptive seeding for supervoxel segmentation. SSV uses visual saliency to guide the process of supervoxel generation. This results in densely distributed, small, and precise supervoxels in salient regions that often contain objects. In less salient regions that often correspond to the background, the supervoxels are larger. Experiments show that this approach improves the quality of the resulting supervoxel segmentation.

Finally, we test our approaches in an in-hand object pose estimation task. The implementation process shows that it is easy and straightforward to adapt CloudSyn to generate suitable training data for the in-hand object pose estimation task. Moreover, the low cost of data generation time enables fast experimental iterations, which is an essential advantage for robotic applications.

Zusammenfassung

Das Ziel der 6D-Objektposenschätzung (engl. 6D object pose estimation) ist es, die 3D-Rotation und 3D-Translation zwischen einem bekannten Objektkoordinatensystem und einem Referenzkoordinatensystem zu finden. Dies ist ein wichtiges Forschungsthema im Bereich der Computer Vision, denn die Kenntnis der 6D-Pose eines Objekts ist eine wichtige Voraussetzung für viele Robotikanwendungen wie das Greifen und die geschickte Manipulation (engl. dexterous manipulation). Die Mehrheit der existierenden Methoden verlässt sich hauptsächlich auf Farbe als primäre Datenmodalität. Sie betrachten die Tiefeninformation entweder als Zusatzinformation oder ignorieren sie komplett. Jedoch enthält die Tiefeninformation geometrische Informationen über die Objektoberfläche, was eine vielversprechende Datenmodalität für die 6D-Objektposenschätzung sein sollte. In dieser Arbeit wird daher untersucht, welchen Beitrag Tiefeninformationen zum Problem der 6D-Objektposenschätzung leisten können. Wir konzentrieren uns dabei auf die Entwicklung von lernbasierten Ansätzen zur 6D-Objektposenschätzung, indem wir nur die Tiefeninformationen verwenden.

Der erste Ansatz, den wir vorstellen, ist CloudPose. CloudPose ist ein auf Deep Learning basierendes System, das aus Tiefeninformationen, die durch Punktwolken repräsentiert werden, 6D-Objektposen schätzt. Wir verwenden Punktwolken als Eingabe sowohl für tiefe Netzwerke als auch für die geometriebasierte Posenverfeinerung. CloudPose verwendet separate Netzwerke für Rotations- und Translationsregression. Wir argumentieren, dass die Darstellung mittels Achse und Winkel eine geeignete Rotationsdarstellung für Deep Learning ist, und verwenden eine geodätische Verlustfunktion für die Rotationsregression. Weiterführende Analysen zeigen, dass diese Designentscheidungen Alternativen wie die Quaternion-Darstellung und L2-Verlust oder die Regression von Translation und Rotation mit demselben Netzwerk übertreffen. Obwohl die Struktur von CloudPose einfach ist, haben experimentelle Ergebnisse konkurrenzfähige Ergebnisse auf öffentlich zugänglichen Datensätzen gezeigt.

Ein Nachteil von CloudPose ist, dass es nicht robust gegenüber durch Rauschen gestörte Eingabedaten ist. Um diesen Nachteil zu überwinden, wird ein weiteres System namens CloudAAE vorgeschlagen. CloudAAE ist ein Autoencoder-basierter 6D-Objektposenregressor, der besonders robust gegenüber dem Rauschen der Tiefeneingabedaten ist. Der Autoencoder lernt einen komprimierten Vektor, der die 6D-Objektpositionsdaten kodiert und invariant gegenüber dem Rauschen in den Eingabedaten ist. Dann wird der komprimierte Vektor als Eingabe für zwei separate Netzwerke zur Rotations- und Translationsregression verwendet. Im Vergleich zu CloudPose hat CloudAAE eine bessere Leistung bei Datensätzen mit durch Rauschen gestörte Tiefendaten.

Sowohl CloudPose als auch CloudAAE sind auf überwachtem Lernen basierende Systeme und benötigen eine große Menge an annotierten Trainingsdaten. Es ist oft gewünscht, auf synthetischen Daten zu trainieren, da die Kosten für die manuelle Annotation von 6D-Objektposen auf realen Daten sehr hoch sind. Es ist jedoch oft teuer, RGB-Bilder zu synthetisieren, da die visuelle Diskrepanz zwischen syn-

thetischen und realen RGB-Bildern sehr groß ist. Im Gegensatz dazu ist diese visuelle Diskrepanz bei Tiefeninformationen wesentlich kleiner und einfacher zu überwinden. Daher präsentieren wir eine leichtgewichtige Datensynthese-Pipeline namens CloudSyn, die synthetische Punktwolkensegmente für das Training erzeugt. CloudSyn benötigt nur texturlose 3D-Objektmodelle und die gewünschten Kamerapositionen und ist sowohl hinsichtlich der Rechenzeit als auch des Speicherplatzverbrauchs der erzeugten Daten kostengünstig. Unser Datensyntheseprozess ist um drei Größenordnungen schneller als die üblicherweise verwendeten Ansätze, die RGB-Bilddaten rendern.

Bei der Evaluierung von CloudPose und CloudAAE haben wir festgestellt, dass ein präzises Segment ohne Hintergrundrauschen in unseren Experimenten oft zu einer genaueren 6D-Positionsschätzung führt. Mit anderen Worten: Die Qualität der Objektsegmentierung hat einen Einfluss auf die Leistung des Gesamtsystems. Zu diesem Zweck präsentieren wir eine Übersegmentierungsmethode für Punktwolken namens SSV. SSV nutzt die visuelle Salienz, um den Prozess der Generierung von Supervoxeln zu steuern. Das Ergebnis sind dicht verteilte, kleine und präzise Supervoxel in auffälligen Regionen, die oft Objekte enthalten. In weniger salienten Regionen, die oft dem Hintergrund entsprechen, sind die Supervoxel größer. Experimente zeigen, dass dieser Ansatz die Qualität der resultierenden Supervoxel-Segmentierung verbessert.

Schließlich testen wir unsere Ansätze in einer Aufgabe zur Schätzung der In-Hand-Objektpose. Der Implementierungsprozess zeigt, dass es einfach und unkompliziert ist, CloudSyn so anzupassen, dass es geeignete Trainingsdaten für die In-Hand-Objektposenschätzung erzeugt. Darüber hinaus ermöglicht der geringe Zeitaufwand für die Datengenerierung schnelle experimentelle Iterationen, was ein wesentlicher Vorteil für Robotikanwendungen ist.

Acknowledgements

First, I want to express my gratitude to my supervisors, Prof. Simone Frintrop and Prof. Jianwei Zhang, who provided me with the opportunity to start a doctoral program. Both of them supported me with valuable guidance, discussion, feedback, and resources that I needed to navigate through my research. I really appreciated their warm encouragement during the ups and solid trust during the downs. I am also grateful to Prof. Markus Vincze, who kindly took on reviewing this thesis.

Second, I want to thank Dr. Mikko Lauri for his constant help and support. I really enjoyed the fruitful discussions we had throughout the years and all the interesting ideas that came out of the discussions. Mikko not only taught me how to write and code but also taught me how to enjoy a good cup of coffee. I want to thank Noha Sarhan for being the kindest office mate, who gave me a warm welcome on my first day and constant positive energy in the office. I want to thank Christian Wilms for his help and support during the tough times and his invitations to various sports events that really enriched my leisure time. I want to thank Kerstin Diop-Nickel and Dieter Jessen for their timely support and help that ensured everything is running smoothly in the group.

Third, I want to thank Tatjana Lu Tetsis and Wiebke Noeske for their kind and constant help with all those complicated but important paperwork throughout the years. I want to thank Dr. Norman Hendrich, Michael Görner, Philipp Ruppel for many interesting discussions and taught me things about robotics. I want to thank Dr. Chao Zeng for his help in proofreading my writings and gave me valuable feedback. I want to thank Shuang Li, Hongzhuo Liang, Di Zhang, Jianzhi Lyu, and Lin Cong for the good times we spend together at work and outside of work.

I gratefully acknowledge the support of the German Science Foundation (DFG) and National Science Foundation of China (NSFC) in project Crossmodal Learning, TRR 169. This project allowed me to work with Prof. Xiaolin Hu and Yulong Wang, which is a very nice learning experience. Also, I want to thank Dr. Di Fu for making my life in Hamburg very interesting and memorable.

I want to thank my parents for their constant love and support, and sometimes indulgence. My parents made it possible for me to travel the world, experience life, start this thesis work and finish this thesis work. I want to thank Xixi for the constant love, comfort and support. Special thanks goes to Tongtong Gao, our family poodle, who always brings good energy to the family and brings us closer together.

List of Publications

Here is the list of publications included in this thesis:

- Ge Gao, Mikko Lauri, Xiaolin Hu, Jianwei Zhang, Simone Frintrop: CloudAAE: Learning 6D Object Pose Regression with On-line Data Synthesis on Point Clouds, Proceeding of International Conference on Robotics and Automation (ICRA), 2021 [35]

- Ge Gao, Mikko Lauri, Yulong Wang, Xiaolin Hu, Jianwei Zhang, Simone Frintrop: 6D Object Pose Regression via Supervised Learning on Point Clouds, Proceeding of International Conference on Robotics and Automation (ICRA), 2020 [36]

- Ge Gao, Mikko Lauri, Jianwei Zhang, Simone Frintrop: Occlusion Resistant Object Rotation Regression from Point Cloud Segments, Proceeding of the ECCV Workshop on Recovering 6D Object Pose, 2018 [38]

- Ge Gao, Mikko Lauri, Jianwei Zhang, Simone Frintrop: Saliency-guided Adaptive Seeding for Supervoxel Segmentation, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017 [37]

Here is a publication that was generated during the period of PhD but is not included in this thesis:

- Zhen Deng, Ge Gao, Simone Frintrop, Fuchun Sun, Changshui Zhang, Jianwei Zhang: Attention based visual analysis for fast grasp planning with a multi-fingered robotic hand, in Frontiers in Neurorobotics, 2019, DOI: 10.3389/fnbot.2019.00060

Contents

List of Figures

List of Tables

Chapter 1

Introduction to 6D Object Pose Estimation

1.1 Background and Motivation

Let's fast forward to the year 2050, imagine you are waiting to see whether your newly purchased household robot will bring you the morning coffee smoothly. The hot coffee is on the kitchen counter and ready to be picked up. As depicted in Figure 1.1, the robot is standing by the kitchen counter, looking down at the coffee through its digital sensors. It is time to pick up the coffee with its robotic hand. Although we humans need little to think about where the coffee is or which part of the coffee cup we should reach for, this is not the case for your robot.

To avoid knocking the cup over and spilling the hot coffee or mistakenly picking up the banana next to it, your robot needs to use a series of abilities in its artificial intelligence package. First, from the captured sensor data of the coffee on the kitchen counter, the robot needs to decide which part of the sensor input belongs to the coffee. Second, after figuring out where the coffee is, the robot needs to figure out its position. After knowing the cup handle's position and orientation, the robot starts to plan how it should move its robotic hand for a successful grasp.

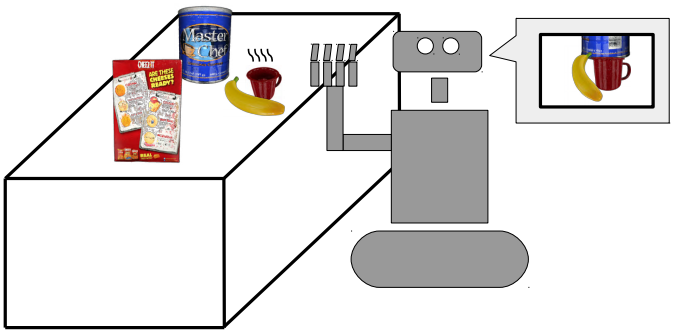


Figure 1.1: The morning coffee scenario.

Overall, the robot is expected to understand its surroundings and interact with the environment without making a mess or harming anyone.

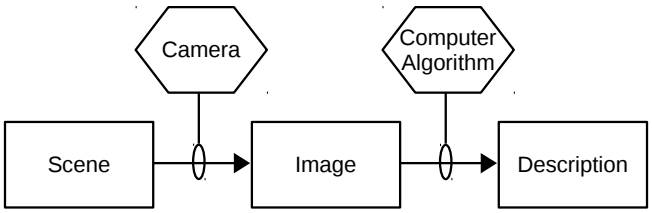
Thirty years ago, Moravec gave a concise description of this scenario [93]: "It is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility." In line with Moravec's paradox, although "locate and pick up the coffee cup" are very trivial tasks for humans, they are among the most challenging tasks in the field of computer vision and artificial intelligence [31]. Despite many advances in the field during the past thirty years, the challenge still persists. Many factors contribute to the challenge. For instance, with the traditional digital camera being the dominant visual sensor for robotic systems, the visual inputs are essentially large matrices filled with numbers. Specifically, a color image with video graphics array (VGA) resolution contains three matrices with 480×640 dimension, and it has almost 1 million values [78]. To understand the environment, information on different semantic levels needs to be extracted from those large matrices. For example, the information could be "this pixel color is yellow", "those pixels form a banana", or "those pixels form a banana, and the banana is edible." Similarly, in order for your robot to interact with the scene, patterns like "the coffee mug is 1.5 meters away and rotated 30 degrees along its up-right axis" need to be extracted. The research areas that deal with this kind of questions are computer and robot vision.

In this thesis, we study a research topic named "6D object pose estimation", a topic emerged from computer vision and which is viewed as an essential prerequirement for many robotic tasks. We investigate this problem from the joint perspective of both computer vision and robot vision. This joint perspective is presented by developing and evaluating algorithms on computer vision benchmarks, as well as applying them to an application related to real-world robotics. We introduce background information such as the concept of computer vision and robot vision, and the challenges of 6D object pose estimation are presented in Section 1.1. The thesis contribution is presented in Section 1.2. The thesis structure is presented in Section 1.3 with figure illustration.

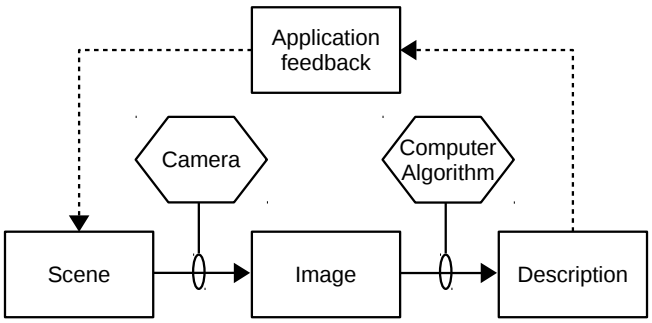
1.1.1 Computer Vision and Robot Vision

The human visual system is extremely good at extracting information from the visual input. Without direct physical contact with the surroundings, vision provides us with a remarkable amount of information about our environment [60]. Since the MIT "Summer Vision Project"[1], which was organized by Seymour Papert [96] in 1966, generations of researchers invested enormous effort into the field of computer vision. They have been developing mathematical algorithms that can do similar tasks as the human visual system [120].

[1]The project expected a group of 11 student workers (including a student coordinator) to develop a robust object detection system during July and August in 1966. Rumour has it that the student coordinator Gerald Sussman never worked in computer vision again after this project.



(a) computer vision system

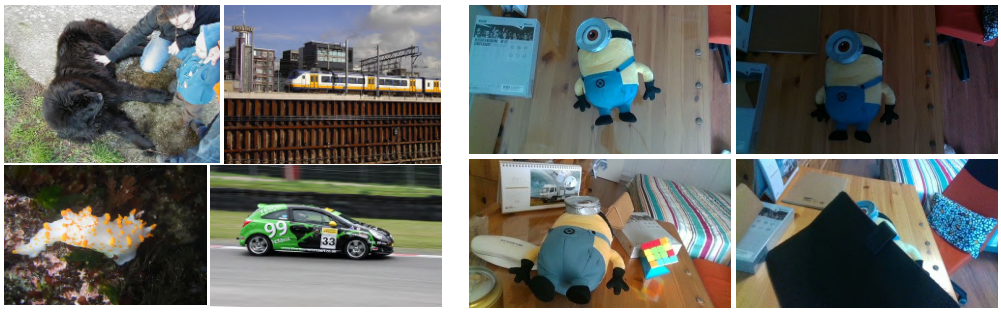


(b) robot vision system

Figure 1.2: Illustration of (a) a computer vision system and (b) a robot vision system. Adapted from [60].

The task of a computer vision system is to extract information from images. Robot vision can be viewed as a subcategory of computer vision. It incorporates aspects of robotics into the algorithms, for example a robot can physically affect and interact with its environment. Figure 1.2(a) is an illustration of a computer vision system. An image is taken by a camera from a scene, and a computer algorithm is used to extract descriptive information from the image. The information can be which objects the image contains or what the cat in the image is doing. In comparison, a robot vision system contains more components [60]. Figure 1.2(b) is an illustration. A robot vision system is a combination of camera hardware and computer algorithms. This allows robots to process visual data and extract the information needed for actions in the real world. Furthermore, the interaction between the robot and its environment may cause changes in the scene, which is relevant to the next action.

Another difference worth noting between computer vision and robot vision is benchmarking images. Figure 1.3(a) shows some example images from the ImageNet dataset [22], which is often used for developing computer vision algorithms. Figure 1.3(b) shows some examples from the OpenLORIS-Object dataset [112], which is a dataset used for teaching robots about life long learning. Observing those two image sets, we can spot three main differences. First, the image illumination quality is generally better in the ImageNet dataset. This is mainly due to that the human photographer tends to optimize the lighting condition before taking an image, while a robotic system may block the light source and take an underexposed photo (Figure 1.3(b), upper left). Second, the imaging angles are generally better in the ImageNet dataset. This is caused again by the human pho-



(a) (b)

Figure 1.3: Example images for a computer vision benchmark and a robot vision benchmark. (a) Examples for a computer vision system, taken from the ImageNet dataset [22]. (b) Examples for a robot vision system, taken from the OpenLORIS-Object [112].

tographer, who tends to find a better angle to capture an appealing aesthetic image of the target scene. It is not hard to understand why this is a very challenging task for robots. Third, the scenarios are more diverse in the ImageNet dataset, while it is more restricted in the OpenLORIS-Object dataset. This is because robotic applications are usually collected from a real-life setup, such as a household or a lab environment.

Overall, robotic applications deal with images that have a wider illumination range and potentially less ideal imaging angles. Although the scenarios are less diverse for robotic vision than its counterpart in the computer vision dataset, it does not mean that robot vision is easier. To operate in real-life scenarios, a robot vision system should be robust to changes in its operating environment. Since 2012, when Alex Krizhevsky and his colleagues won the ImageNet [22] challenge with a deep convolutional neural network (CNN) [73], deep learning-based approaches start to dominate vision-related research areas. One constraint of deep learning is that for a task, the training and testing data should be drawn from the same probability distribution [41]. Hence, this low diversity in datasets limits the applicability of the system in real scenarios. For example, if a system is trained on data collected from one apartment, it can not be guaranteed that this system works in a different apartment with different lighting conditions and objects. In general, it is desired to design a system that can generalize well or have a good ability to scale up to many objects or environments. It is also important to find a balance between the speed and cost of the robot vision algorithm. Furthermore, the accuracy and robustness need to be guaranteed for good robot-environment physical interactions.

Apart from benchmarking images, there are also other differences in a broader scope. For example, robots can be equipped with multiple vision sensors and the sensory inputs can be combined for information extraction. This can be very useful for many applications but also arise challenges such as sensor calibration and real-time processing. Since robots can take actions upon perception, they can also actively change their positions in the environment to obtain a better viewpoint.

This active perspective of perception is beneficial in many robotic tasks, and it also has challenges such as how to decide where to move for the next step. Those differences are not in the scope of this thesis and will not be investigated here.

The task of 6D object estimation is a classical topic in computer vision and an important task in the field of robot vision. In this thesis, we focus on developing 6D object pose estimation systems that are lightweight and easily deployable on a robotic system. Ideally, our methods should be accurate, fast, and can easily scale up to a large number of objects.

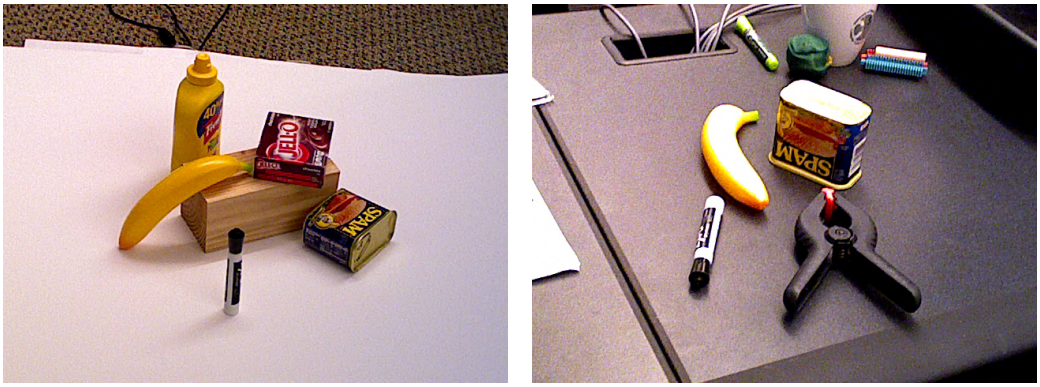
1.1.2 Challenges of 6D Object Pose Estimation

In our morning coffee example, the process of deciding how the mug is placed is the task of 3 degrees of freedom (3D) object detection, or more recently known as 6D object pose estimation. A reasonably precise definition of the 6D object pose estimation problem appeared as early as 1985, and it is proposed by Besl and Jain [9]. According to Besl and Jain, if given a known coordinate system, "the coffee mug is 1.5 meters away" are the 3D location or translation parameters of the mug. Moreover,"the mug is rotated 30 degrees along its up-right axis" is the 3D orientation or rotation parameters.

After more than 30 years of investigation, several challenges remain to be solved for this topic. There are four main challenges. For systems that rely on color information, the first challenge is illumination change. Slight changes in the illumination condition can cause drastic changes in the red, green, blue (RGB) values of a pixel even when it is not detectable by the human vision. Figure 1.4 shows an example of how the change of illumination condition impacts the object's appearance in the YCB Video dataset [139]. Both images contain the same banana, the right image is overexposed and the banana appears white instead of yellow. This kind of scenario makes developing a robust system very challenging. Moreover, shadows from other objects in the environment or the object's reflective property also increase the challenge.

Another challenge is occlusion, either self-occlusion or external occlusion. While it is relatively easy to estimate the 6D pose when the object is fully observable, missing parts of the object can cause the system to not work or provide inaccurate results. Figure 1.5 shows examples of different kinds of occlusion. The self-occlusion example (Figure 1.5(b)) illustrates how the black part of the drill head is occluded by the orange part of the drill head. To detect the 6D pose of the power drill without occlusion (Figure 1.5(a)) is easier than estimating in the case of self-occlusion (Figure 1.5(b)) or external occlusion (Figure 1.5(c)).

The third challenge is the pose ambiguity. Most human-made objects have certain degrees of rotational symmetry, such as a cubic or cylinder shape. It can appear that although having the same visual appearance, the defined pose in its object coordinate system is different, which presents ambiguity. Figure 1.6 shows a rotational symmetric object, a foam brick. Although both images are very similar visually, there is a 180° difference around the upper right axis. The difference in viewpoint can be seen from the subtle differences on the brick's left-side edges

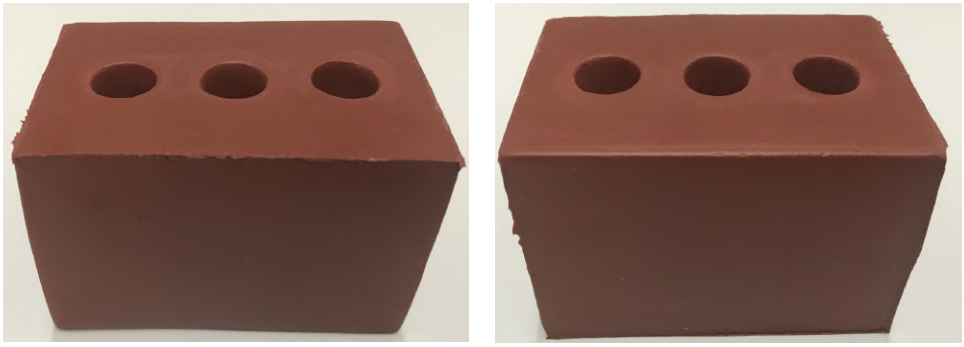


(a) Normal (b) Overexposed

Figure 1.4: The same banana appears to be in different colors due to different illumination conditions. Images are taken from the YCB Video dataset [139].

(a) No occlusion (b) Self-occlusion (c) External occlusion

Figure 1.5: Examples of different kinds of occlusion of a power drill.



(a) View 1 (b) View 2

Figure 1.6: An example of rotational symmetry. These two images containing the same foam brick look very similar. However, they are taken from two different viewpoints, and this difference can be seen from the subtle differences on the brick's left-side edges.

in the images. Furthermore, objects such as a coffee mug are almost symmetric except for the handle, and when the handle is not observable due to occlusion, the problem of ambiguity arises.

The fourth challenge is from the practical perspective. Among the state-of-the-art deep learning-based approaches, it is common to have an individually trained network for each target object, which is not practical if there are hundreds of different objects. Also, as mentioned above, the existing datasets have a low diversity of objects and environments, limiting the applicability of the system in real scenarios. One reason for this low diversity is that the data collection process is often expensive in terms of time and other resources such as human resources and hardware storage.

The first three challenges impact the system performance, while the fourth challenge impacts the difficulty level of applying a pose estimation system in real-world scenarios, and they are all open research topics. Since we focus on using depth information in this thesis, our systems are not impacted by the first challenge. The second challenge is normally handled by introducing occlusions into the training data [123]. The third challenge is investigated with various approaches [21, 100] and it is not in the scope of this thesis. In this thesis, we mainly focus on the fourth challenge while having an accurate system. We focus on developing a multi-class system, i.e., one system that can handle all the objects in a dataset. We also focus on decreasing the difficulty level of integrating our pose estimation system into real-world applications.

1.1.3 Thesis Focus

As mentioned before, robot vision systems consist of cameras and algorithms, the cameras provide input image data for the system and the algorithm converts the input data into desired information. There is more than one design choice for each component, and each choice leads to a different research focus for the task of 6D object pose estimation. In this section, we briefly discuss the options for each component and describe the focus of this thesis.

Data modality. First of all, the two main data modalities for robotic visual perception are color and depth. Color contains information such as RGB-valued color and texture, and depth contains surface geometry information. Texture information is very useful for capturing image details and recognizing, for example, hand writings [77] or human faces [129]. However, color information is susceptible to illumination changes, and it is very challenging to make a method robust against a big range of illumination changes. On the other hand, surface geometry is very useful for extracting shape information such as planes and cylinders [108], but it lacks detailed texture information. In general, those two modalities have their advantages and disadvantages and complement each other well in a multi-modality setting. For the task of 6D object pose estimation, many deep learning-based methods have explored how to estimate accurate 6D object pose using color information [68, 14, 94]. Among other works that consider both color and depth information [139], depth is

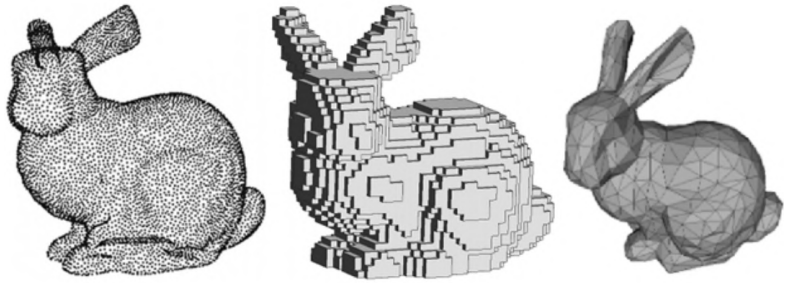


Figure 1.7: 3D data representations: point cloud (left), voxel grid (middle), mesh (right). Adapted from [54].

often used as a secondary modality for pose refinement. In general, a lot of existing methods concentrate on using color, and the depth is understudied. Therefore, to fill this gap, we focus on depth information and investigate what depth can contribute to the task of 6D object pose estimation in this thesis.

Data representation. A suitable representation is needed for encoding depth information. Depth information can be represented in an either 2.5D or 3D form. The 2.5D representation is also known as the depth map. A depth map is essentially a 2D image, and each pixel contains the distance information between the surface point and the image plane. This kind of data can be easily obtained from commercially available depth sensors such as the Microsoft Kinect[2]. One drawback of this representation is that it is not a full 3D representation since it only contains distance information.

Regarding full 3D representations, unlike 2D images with a dominant 2D pixel grid representation, 3D data can be represented by many commonly used formats such as voxel grids, meshes, and point clouds. Figure 1.7 shows an illustration of the Standford bunny represented in a point cloud, a voxel grid, and a triangle mesh. Since the computational cost for 3D data is very high due to the extra dimension, it is important to choose a less expensive 3D representation. A voxel representation is very simple but bulky. It contains many empty spaces, and the quantization of the target space unavoidably leads to quantization artifacts and loss of information. A 3D mesh is more efficient by only representing the object surface, but it is a complicated representation containing vertices, edges, and faces. On the other hand, a point cloud contains the full information from sensor input and it is a very simple representation [101]. Hence, point clouds are computationally very cheap and still encode the object shape information. Therefore, in this thesis, we focus on developing 6D pose estimation systems that operate on point clouds.

Methodology. Before starting this thesis work, deep learning has gained its popularity in many visual recognition tasks. Compared to hand-crafted features, the deep learned features have largely improved benchmark scores in many tasks [22].

[2]https://developer.microsoft.com/de-de/windows/kinect/

This makes it intriguing to use deep learning methods in this thesis. On the other hand, researchers still hold onto traditional non-deep learning approaches due to their transparency and robustness. In this thesis, we mainly focus on using deep learning approaches on depth data to handle the task of 6D pose estimation. Meanwhile, we also combine deep learning approaches with traditional methods for achieving better system performance.

1.2 Thesis Contributions

In this thesis work, we contribute to the field of 6D object pose estimation using depth information represented by point clouds. Figure 1.8 illustrates the connections among the contributions. Here we present a summary of the main contributions:

- Introduction of the first supervised learning-based system that regresses 6D poses from point clouds. We adapt an existing method for deep learning on point clouds as the system backbone and design a simple system for 6D pose regression on point clouds. Using only depth data during the pose inference stage, our system outperforms comparison methods that use both color and depth information for pose inference. We name this work **CloudPose** (Chapter 4, also published in [38, 36]).

- One drawback of CloudPose is that it is not robust against noise in the input data. Hence, we introduce a hybrid system that combines self-supervised learning and supervised learning. We use an augmented autoencoder to implicitly learn a latent representation that encodes the 6D pose information. Meanwhile, we directly regress 6D object poses from this latent vector in a supervised learning setup. Experimental results show that there are both advantages and disadvantages of this work. We name this work **CloudAAE** (Chapter 5, also published in [35]).

- To generate training data more efficiently, we present a point cloud-based online data synthesis pipeline that enables generating synthetic data in a simple and cheap manner. Our pipeline is lightweight, and the cost is cheap in terms of time and hardware storage. Moreover, this pipeline also enables agile deployment of the 6D pose estimation system in robotic applications. The efficiency of this pipeline is verified with public benchmarks. We name this work **CloudSyn** (Chapter 6, also published in [35]).

- For CloudPose and CloudAAE, the accuracy of object segmentation affects the accuracy of 6D pose estimation. When using color-based semantic segmentation, the resulting object segments sometimes contain pixels that are physically far away from the object. To improve the segmentation quality, one potential idea is to integrate the results from a depth-based segmentation. Towards this end, we propose a point cloud-based over-segmentation method that uses color saliency to guide the initial seeding process. Instead

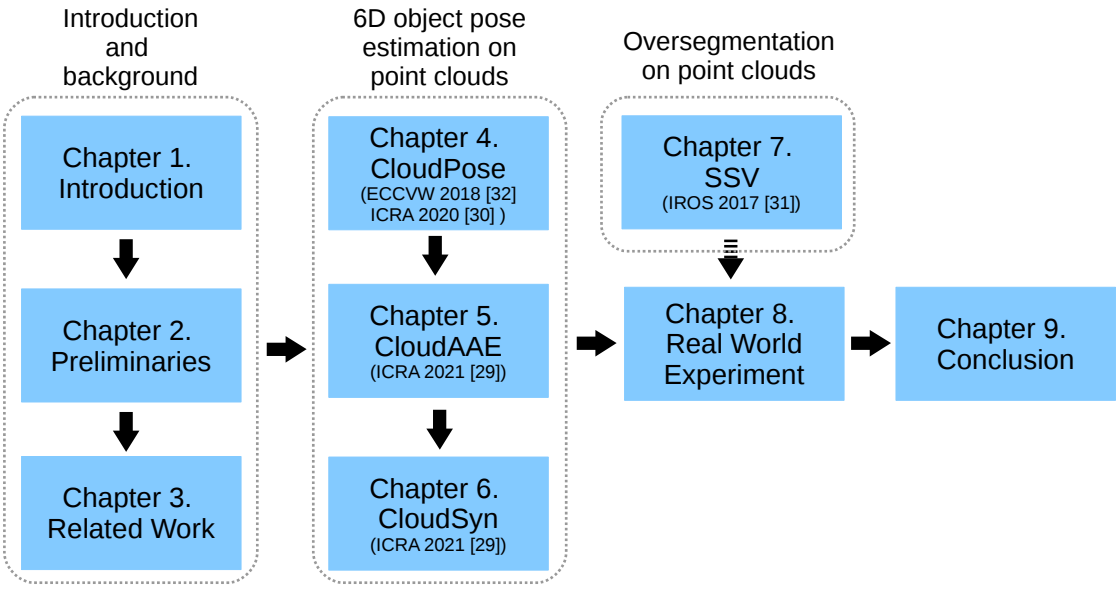


Figure 1.8: The structure of this thesis.

of uniform seeding, dense seeds are initialized in salient areas, and less dense seeds are initialized in non-salient regions. This adaptive seeding helps the system to preserve the object boundaries better. We name this work **SSV** (chapter 7, also published in [37]).

- A case study in real-world experiments that examines the proposed 6D object pose estimation systems in an in-hand object pose estimation task. By testing our systems in a real-world setup, we verify our system's usability and gain insights about the important aspects when integrating a vision system with a real-world setup (chapter 8).

1.3 Thesis Structure

The thesis structure is illustrated in Figure 1.8. The remainder of the thesis is structured into eight chapters. Chapter 2 introduces the preliminary concepts and Chapter 3 presents the state of the art of 6D object pose estimation. The following five chapters present each contribution of this work:

- Chapter 4 introduces CloudPose, a simple and effective 6D object pose estimation system that operates on point cloud segments. The focus of this work is proposing a system that can produce accurate 6D pose estimates using depth information. We describe the important factors and design choices.
- Chapter 5 introduces CloudAAE, which is another variant of point cloud-based 6D object pose estimator. The main focus is to adapt the idea of an augmented autoencoder for a point cloud system, which improves the performance and robustness of the system.

- Chapter 6 introduces CloudSyn, which is an on-line data synthesis pipeline. The main advantage of our pipeline is its low cost in the sense of time and hardware storage. Extensive ablation studies are investigated for pipeline components.

- Chapter 7 introduces the point cloud-based over-segmentation work named SSV. Visual saliency is used to guide the initial seeding process, and this adaptive seeding helps boost the system performance.

- Chapter 8 examines the applicability of our developed system to a real-world scenario. The main focus is to connect the data synthesis pipeline and the 6D object pose estimator with a real-world workspace setup.

Finally, Chapter 9 concludes the thesis. The main contributions are summarized, with discussions on the strengths and limitations. An outlook on future work is also included.

As illustrated in Figure 1.8, Chapters 1 and 2 present the background motivation and basic introduction to the thesis topic, and 3 presents the recent development in the field of 6D object pose estimation. Chapter 4 and 5 present two approaches that learn 6D pose from point cloud segments. Chapter 6 presents a pipeline that synthesizes point cloud segments for training a 6D pose estimation network. Chapter 7 presents a work in the field of over-segmenting point clouds. Combining the chapters on 6D pose estimation (5, 6), we conduct a real-world experiment in Chapter 8. Theoretically, the over-segmentation approach in Chapter 7 can potentially help to improve the performance of the real-world experiments. However, it is not integrated into the real-world experiments in this thesis. The main reason is that it is practically easier to use the recently developed semantic segmentation methods for object segmentation.

Chapter 2

Preliminaries to Learning 6D Object Pose

In this chapter, we present a bundle of preliminaries that are relevant to this thesis. First, the problem formulation of 6D object estimation is presented in Section 2.1. The 3D rotation group and rotation representation are introduced in Section 2.2. Section 2.3 presents how to conduct deep learning on unorganized data such as point clouds. Finally, the commonly used datasets and evaluation metrics are introduced in Section 2.4 and 2.5, respectively.

2.1 Problem Formulation

In this section, we provide a formal description of the problem that we address in this thesis. We describe the input and output of a 6D pose estimation system. We also describe the training and testing setup for a learning-based system.

2.1.1 6D Pose Estimation

In this thesis, we deal with the problem of estimating the 6D pose of known objects, represented in a 3D point cloud. The 6D pose of an object is composed of the 3D location $\mathbf{t}$ and the 3D orientation R. A 6D pose describes the transformation from a local coordinate system of the object to a camera coordinate system. Figure 2.1 shows an illustration. We denote the local coordinate system with O and the camera coordinate system with C.

We denote the point cloud object model with

$$P^O = \left\{ \mathbf{x}_i^O \in \mathbb{R}^3 \mid i = 1, \ldots, m \right\}, \tag{2.1}$$

where $\mathbf{x}_i^O$ is the ith point, and m is the number of points in the object model. Figure 2.2(a) shows an illustration of an object model. The creator of the respective dataset manually defines the position of the object coordinate system. For the target object viewed by the camera, we denote the object in the camera view with

$$P^C = \left\{ \mathbf{x}_i^C \in \mathbb{R}^3 \mid i = 1, \ldots, n \right\}, \tag{2.2}$$

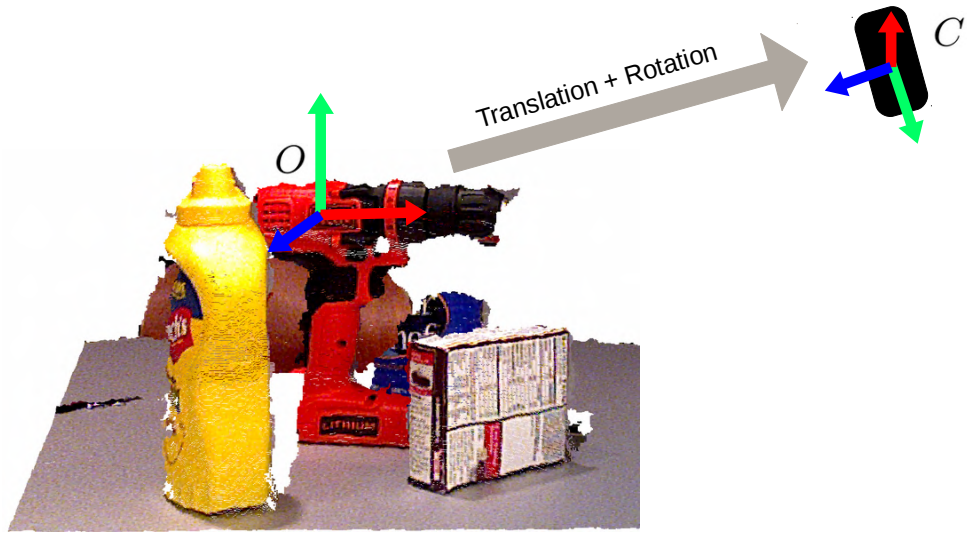


Figure 2.1: The goal of 6D pose estimation is to find the translation and rotation from the object coordinate frame O to the camera coordinate frame C.

(a) (b)

Figure 2.2: Illustration of the object model and object segment in the camera view. (a) The object model P^O at object coordinate O. (b) An object segment P^C in the camera view C.

where $\mathbf{x}_i^C$ is the ith point, and n is the number of points in the object segment. Figure 2.2(b) shows the illustration of an object segment.

Given a set of points P^C on the surface of a known object in the camera coordinate, the aim of pose estimation is to find a 6D transformation that transforms x_i from the object coordinates to the camera coordinates

$$\mathbf{x}_i^C = R\mathbf{x}_i^O + \mathbf{t}, \tag{2.3}$$

where $\mathbf{x}_i^C$ and $\mathbf{x}_i^O$ are the corresponding points in the object segment and object model, respectively. The commonly used unit for measuring the amount of 3D rotation is radian or degree, and the unit for measuring the amount of 3D translation is meter.

2.1.2 Learning 6D Pose Estimation

Hodaň [56] et al. provide a formal definition of learning 6D object pose estimation. In this section, we adapt their formulation and add more descriptions regarding our scenario with point clouds.

For a learning-based 6D object pose estimator, there is a training phase and a testing phase. At training time, the system is provided with a training set containing a set $\mathbf{T}$ of known rigid objects

$$\mathbf{T} = \{T_1, T_2, \ldots, T_k\}, \tag{2.4}$$

and the known objects are denoted with $O = \{1, 2, \ldots, k\}$. k is the number of object classes. In general, the training data T_i can be created synthetically with 3D object models or a set of real RGB or RGB-D images. Meanwhile, the ground truth 6D object poses for each object in each image are also provided.

At the test time, the system is provided with a single test RGB or RGB-D image I and the goal is to estimate the 6D pose for each visible object instance in I. Image I contains a list of objects $L_I = \{o_1, o_2, \ldots, o_k\}$, where $o_i \in O$ denotes object classes presented in I. The test output is a sequence

$$E_I = ((o_1, \hat{\mathbf{t}}_1, \hat{R}_1), (o_2, \hat{\mathbf{t}}_2, \hat{R}_2), \ldots, (o_k, \hat{\mathbf{t}}_k, \hat{R}_k)), \tag{2.5}$$

where 3D translation $\hat{\mathbf{t}}_i$ and 3D rotation $\hat{R}_i$ are the estimated 6D pose of object $o_i \in O$.

In this thesis, the input to our pose estimation system T_i is a point cloud object segment, accompanied by object class information. This segment can be obtained either from a 3D object model or through semantic segmentation from an RGB-D image.

2.2 3D Rotation Representation

A 3D rotation describes a transformation that rotates an object around an axis with an angle. In this section, we introduce three 3D rotation representations, i.e., rotation matrix, axis-angle, and quaternion. Then, we will investigate a suitable rotation representation for a supervised learning system.

Rotation matrix In a 3D Cartesian coordinate system, there are three basic rotations. Each basic rotation rotates about one coordinate axis. Rotating about x-, y-, or z-axis by an angle θ can be written as matrix

$$R(\hat{\mathbf{x}}, \theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \tag{2.6}$$

$$R(\hat{\mathbf{y}}, \theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \text{ and} \tag{2.7}$$

$$R(\hat{\mathbf{z}}, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.8}$$

where $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, $\hat{\mathbf{z}}$ denote the unit vectors along x-, y-, or z-axis, respectively.

An arbitrary rotation can be obtained by combining those basic rotations with matrix multiplication. A rotation R that first rotates about z-axis by γ, then rotates about y-axis by β, and finally rotates about x-axis by α can be obtained by

$$R = R(\hat{\mathbf{x}}, \alpha)R(\hat{\mathbf{y}}, \beta)R(\hat{\mathbf{z}}, \gamma), \tag{2.9}$$

which also represents a rotation whose rotation angles are γ, β, α about axes z, y and x, respectively. One constraint for the rotation matrix R is that it is an orthogonal matrix with determinant one [43].

Axis-angle A 3D rotation can also be described with its axis of rotation and its angle of rotation. As illustrated in Figure 2.3, in a 3D Euclidean space, the axis of rotation is denoted with a unit vector $\hat{\mathbf{e}} \in \mathbb{R}^3$, and the magnitude of rotation is denoted with angle $\theta \in \mathbb{R}$. A rotation R defined with $\hat{\mathbf{e}}$ and θ can be written as

$$R(\hat{\mathbf{e}}, \theta) = \theta\hat{\mathbf{e}}. \tag{2.10}$$

Therefore, the rotation $R(\hat{\mathbf{e}}, \theta)$ in the axis-angle representation can be written as $(\theta e_x, \theta e_y, \theta e_z)$. Figure 2.3 shows an illustration of how a rotation is represented in the axis-angle format.

Quaternion Quaternion is developed by W. R. Hamilton [44]. It is a convenient mathematical notation for representing transformations of points in the 3D space. For a standard orthonormal $\mathbb{R}^3$, its three unit vectors are $\hat{\mathbf{i}} = (1, 0, 0)$, $\hat{\mathbf{j}} = (0, 1, 0)$, $\hat{\mathbf{k}} = (0, 0, 1)$ [75]. A quaternion $\mathbf{q}$ is defined as the sum of $q_0 \in \mathbb{R}$ and $\mathbf{q_v} = (q_1, q_2, q_3) \in \mathbb{R}^3$

$$\mathbf{q} = q_0 + \mathbf{q_v} = q_0 + q_1\hat{\mathbf{i}} + q_2\hat{\mathbf{j}} + q_3\hat{\mathbf{k}}, \tag{2.11}$$

where q_0 is called the scalar part and $\mathbf{q_v}$ is called the vector part. The scalars q_0, q_1, q_2, q_3 are called the components of the quaternion. Assuming a rotation is defined with angle θ around a unit vector $\hat{\mathbf{e}}$,

$$\hat{\mathbf{e}} = (e_x, e_y, e_z) = e_x\hat{\mathbf{i}} + e_y\hat{\mathbf{j}} + e_z\hat{\mathbf{k}}, \tag{2.12}$$

then the corresponding quaternion representation $\mathbf{q}$ is

$$\mathbf{q} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(e_x\hat{\mathbf{i}} + e_y\hat{\mathbf{j}} + e_z\hat{\mathbf{k}}). \tag{2.13}$$

Here $\mathbf{q}$ is a unit quaternion with $\|\mathbf{q}\| = 1$, the rotation $R(\hat{\mathbf{e}}, \theta)$ in the quaternion representation can be written as the components $(\cos\frac{\theta}{2}, e_x\sin\frac{\theta}{2}, e_y\sin\frac{\theta}{2}, e_z\sin\frac{\theta}{2})$.

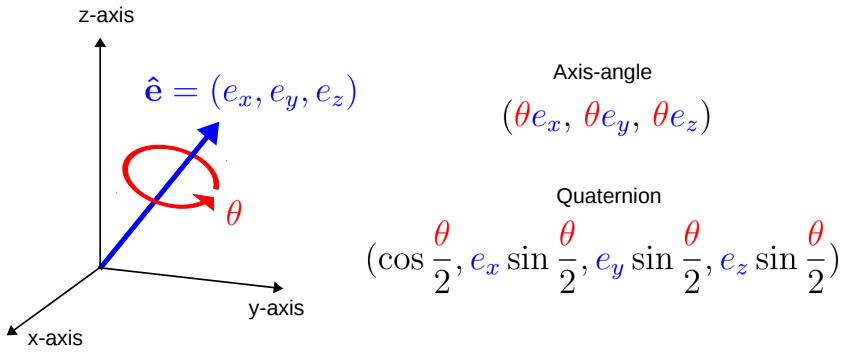


Figure 2.3: Illustration of axis-angle and quaternion.

Figure 2.3 shows an illustration of how a rotation is represented in quaternion format.

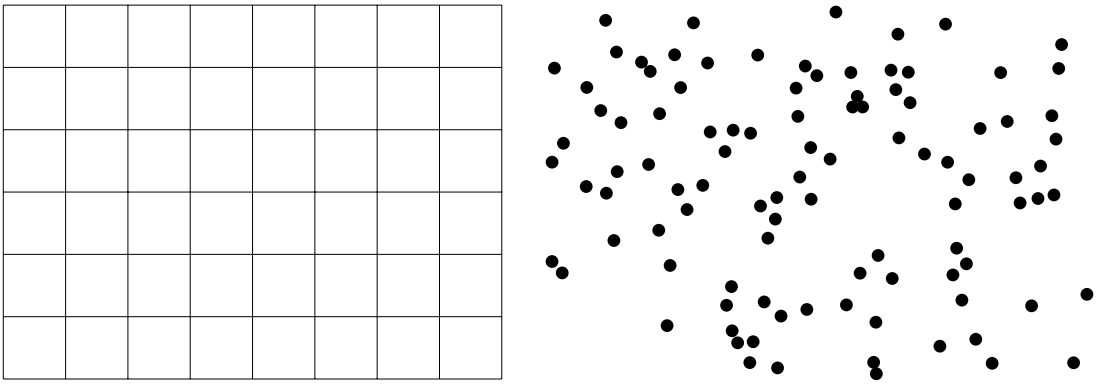
For a learning system, the rotation matrix has 9 parameters, which is less compact compared to the quaternion and axis-angle representation. Furthermore, the rotation matrix is constrained to have a determinant of one, which could potentially complicate the learning task. Similarly, the unit norm constraint also applies to quaternions. Hence, we argue that axis-angle is a suitable rotation representation for a deep learning system.

2.3 Deep Learning on Point Cloud

Since this thesis focuses on learning poses from point clouds, we describe the basic methods for deep learning on point clouds in this section. Before diving into deep learning on point clouds, we briefly touch on the history of deep learning on depth data and motivate deep learning on point clouds.

As mentioned in Section 1.1.3, one common depth representation is the 2.5D depth map. Before deep learning methods on point clouds are proposed, one common way for depth-based deep learning is using CNNs. A CNN is a specialized kind of neural network used to process data with a grid-like structure, for example, an image [41]. CNNs are very powerful at extracting features from color images. Due to their promising performance in various visual recognition tasks, there are also attempts to use CNNs for processing depth information. More specifically, a depth map is often treated as an additional channel to a color image and is used for feature extraction with CNNs [69, 148, 80, 17]. However, depth maps only contain the distance information on one out of three axes, and the complete 3D information is not captured in those depth maps. Hence, it is desired to extract features from point clouds that contain complete 3D information with deep networks.

A point cloud is essentially a set of unordered points, in contrast to an image with a matrix structure. Compared to an image with a grid-like structure, point clouds often appear in an irregular format, which cannot be processed by a system with convolutional architecture [103]. Figure 2.4 shows an illustration of a regular grid-like structure (2.4(a)) and an irregular point set (2.4(b)).



(a) An image with grid-like structure, each square represents one pixel

(b) A point set with an irregular structure

Figure 2.4: Illustration of a regular image grid and an irregular point set.

This section introduces the methods used for deep learning on point clouds in this thesis. Section 2.3.1 introduces the first deep learning structure proposed for the unordered point set, which is called PointNet [103]. Section 2.3.2 introduces a variant of PointNet, which is called PointNet++. Compared to PointNet, PointNet++ enlarges the local receptive field for each point. Section 2.3.3 introduces a system structure that uses the Dynamic Graph (DG). Among those methods, we use PointNet as the building block in our system presented in Chapter 4 and Dynamic Graph in our system presented in Chapter 5.

2.3.1 PointNet

PointNet was proposed by Qi et al. [103] in 2017 and is the first deep learning network for point clouds. It was initially proposed for applications such as object classification and semantic segmentation, and this section uses the object classification task as an example to introduce its feature extraction pipeline. It presents a universal method for extracting deep features from raw point clouds and can be adapted for other tasks.

Qi et al. pointed out three main properties of a point set that represents a point cloud. The first property is the points in the set are unordered, and a network that consumes a 3D point set with N points should be invariant to $N!$ permutations of the input point order. The second property is the interaction among points. Since the points together represent an object surface, they should not be treated isolated and the network should be able to capture both local and global structures. The third property is invariance under transformations. This means if a certain rotation or translation is applied to the point set, the global point cloud category or the semantic segmentation of the points should not be modified.

A point cloud is represented as a set of 3D points $\{\mathbf{p}_i|1,\ldots,n\}$, and $\mathbf{p}_i$ is a vector of its 3D coordinate (x, y, z). For the task of classifying k objects, the network outputs k scores for the k candidate classes. Figure 2.5 illustrates the

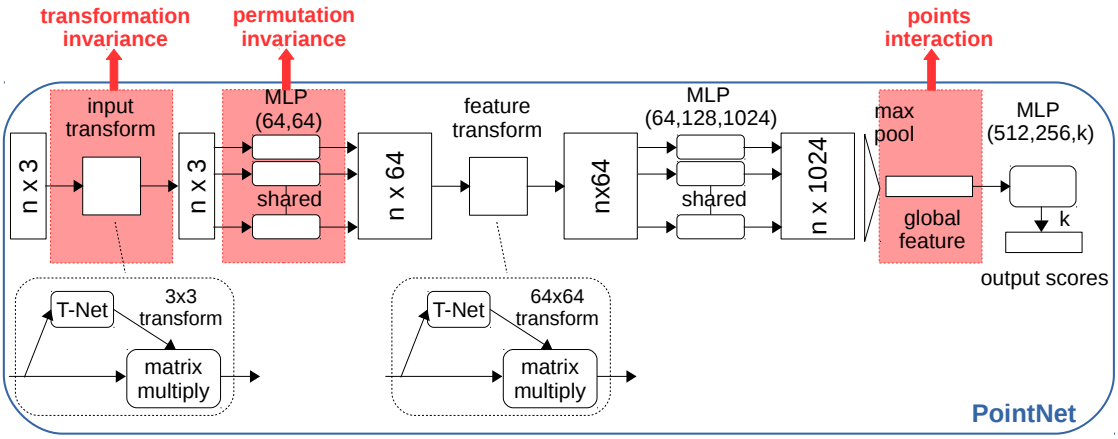


Figure 2.5: PointNet Architecture for object classification. The network takes n points as input and applies input and feature transformations. A mini network (T-net) is used for predicting affine matrices. Each input point is processed individually with MLP for feature extraction. A max-pooling layer is used to obtain the 1024-dimensional global feature, and this feature is used for final class prediction. The red annotations denote the functionality of each main building block. Adapted from [103].

structure of PointNet, and the red annotations denote the functions of the main building blocks, corresponding to the properties mentioned above. The input is a matrix of dimension $n \times 3$, in which n is the number of points in the point set, and 3 corresponds to the 3D coordinate. This input is first processed with a transformation block, which aims to transform the input point set into a canonical pose by applying a 3×3 rotation transformation. This rotation matrix is predicted by a mini network (T-net in Figure 2.5). This transform block is used to achieve the invariance under transformation. To achieve the invariance under point order permutations, all points are processed independently using multi-layer perceptrons (MLP) with shared weights (MLP in Figure 2.5). After a series of feature extraction with MLP layers, a 1024 dimension feature vector is obtained for each point (the $n \times 1024$ block in Figure 2.5). To achieve the interaction among points, those feature vectors are max-pooled to create a global feature representation of the input point cloud. Finally, after some MLP layers, the network outputs a k-dimension vector, which contains the classification scores.

2.3.2 PointNet++

One main drawback of PointNet is that it does not capture local structures [104]. Since PointNet only captures a global representation for the input point set, it has limited ability to recognize fine-grained patterns and to generalize to complex scenes [104]. To improve this aspect of a deep learning structure on point clouds, the authors of PointNet proposed PointNet++ [104].

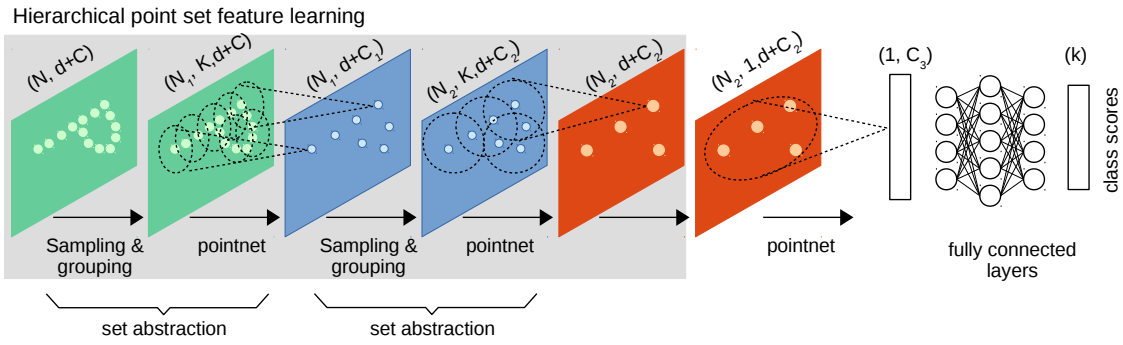


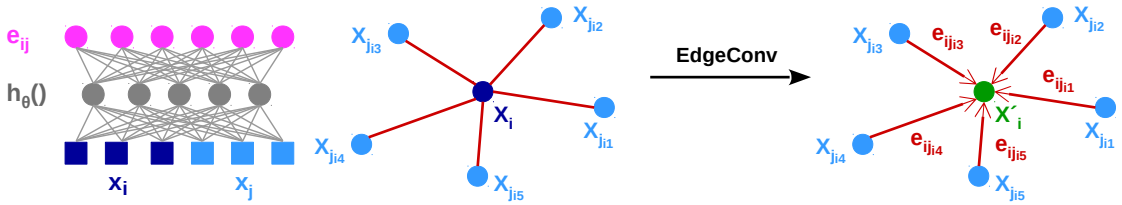
Figure 2.6: Illustration of the hierarchical point set feature learning in PointNet++. Each set abstraction level contains a Sampling layer, a Grouping layer, and a PointNet layer. A set of points is processed and abstracted into a new set with fewer elements after each set abstraction level [104].

In PointNet++, Qi et al. proposed the hierarchical point set feature learning. The general idea is to group points hierarchically, and meanwhile progressively abstract larger and larger local regions [104]. This hierarchical architecture is composed of several set abstraction levels. A set abstraction level contains a sampling layer, a grouping layer, and a PointNet layer. The Sampling layer selects a subset of the input points and defines the subset as the centroids of local regions. The grouping layer constructs local regions for each centroid by finding the near neighbors. The PointNet layer uses a PointNet-like structure to encode each local region into feature vectors.

Figure 2.6 shows an illustration of the hierarchical feature learning architecture. The input is a point set of N points, with d-dimensional coordinates and C-dimensional point features. With farthest point sampling (FPS), the Sampling layer defines N_1 centroids. The Grouping layer finds K nearest neighbors for each centroid and groups each centroid with its corresponding neighbor. For each group, a PointNet layer is applied to extract C_1 features for the group, and one set abstraction level is completed. This level is repeated a desired number of times until a global feature vector is obtained (with dimension $(1, C_3)$ in Figure 2.6). Finally, some fully connected layers are used for obtaining the final class scores.

2.3.3 Dynamic Graph

An alternative method to capture the local structure is the Dynamic Graph proposed by Wang et al. [132]. Instead of extracting point features directly from their embedding, Wang et al. proposed to construct graphs with edge that describe the relationships between a point or a feature and its neighbors [132]. The graph is recomputed using nearest neighbours in the feature spaces computed by each previous layer, hence the system is named Dynamic Graph. The edge convolution operation (EdgeConv) is used to extract features from the edges. The EdgeConv operation also maintains permutation invariance.



(a) Computing edge feature

(b) The EdgeConv operation

Figure 2.7: Illustration of edge feature and edge convolution, adapted from [132]. (a) An edge feature $\mathbf{e}_{ij}$ is computed from the point pair $\mathbf{x}_i$ and $\mathbf{x}_j$. (b) The EdgeConv operation calculates the output by aggregating all the edge features corresponding to each connected vertex.

Assume a p-dimensional set is denoted with $X = \{\mathbf{x}_i \in \mathbb{R}^p \mid i = 1, \ldots, m\}$, in which $\mathbb{R}^p$ can be either a 3D Euclidean space or an arbitrary feature space. For each point $\mathbf{x}_i$, a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is first calculated. The vertices are denoted with $\mathcal{V} = \{1, \ldots, n\}$ and edges are $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. A simple case for this direct graph is the k-nearest neighbor graph. The k-nearest neighbor graph contains directed edges $(i, j_{i1}), \ldots, (i, j_{ik})$, in which $\mathbf{x}_{j_{i1}}, \ldots, \mathbf{x}_{j_{ik}}$ are the k closest points to $\mathbf{x}_i$. The edge feature $\mathbf{e}_{ij}$ is defined as

$$\mathbf{e}_{ij} = h_\Theta(\mathbf{x}_i, \mathbf{x}_j) \tag{2.14}$$

where h_Θ is some parametric non-linear function parameterized by Θ, and Θ is a set of learnable parameters. Figure 2.7(a) shows an illustration.

The edge features are then processed by an edge convolution operation (EdgeConv), which contains an edge function h and an aggregation operation $\square$ [132]. The aggregation operation can be a max pooling or average pooling operation. For the i-th vertex, the output of EdgeConv is

$$\mathbf{x}'_i = \underset{j:(i,j)\in\mathcal{E}}{\square} h_\Theta(\mathbf{x}_i, \mathbf{x}_j) \tag{2.15}$$

where Θ is a set of learnable parameters. In this case, x_i is the center point of the graph, and $\{j : (i, j) \in \mathcal{E}\}$ represents its local neighbor patch. Figure 2.7(b) shows an illustration of the EdgeConv operation.

2.4 Datasets

Many datasets have been proposed for benchmarking 6D object pose estimation [139, 52, 55]. The general goal of those datasets is to provide a joint base to test methods from one or several challenges (mentioned in Section 1.1.2) such as whether the objects are textured, or are rotationally symmetric, or whether the illumination condition varies a lot. With the development of technology and research in this field, the proposed datasets tend to become larger and more challenging.

In this thesis, we choose three datasets that are most commonly used and suit our application for evaluation, namely, the LineMOD (LM), the Occluded LineMOD (LMO), and the YCB video dataset (YCBV). All of the datasets contain a color image and a depth image for each frame, and they are calibrated. The 6D poses of the target object in the corresponding images are annotated. Furthermore, the intrinsic camera parameters are also provided. Throughout this thesis, we use the full names of the datasets and their corresponding abbreviations interchangeably.

2.4.1 LineMOD Dataset

Hinterstoisser et al. proposed the LineMOD dataset in 2012 [52]. It contains 15 objects from daily scenarios containing ape, bench vise, bowl, can, cat, cup, driller, duck, glue, hole puncher, iron, lamp, phone, camera, and egg box. The 15 objects are filmed in 15 videos, with more than $18,000$ frames. The 3D model of each object is provided in both mesh and point cloud format, and the maximum diameters of the objects are also provided. For training and testing the system, 15% of the frames are training data and 85% of the frames are testing data. By convention, only 13 out of 15 objects are used for system evaluation [68]. This is because the object bowl and cup lack meshed models for creating synthetic training data.

The majority of the objects are low textured and do not have rotational symmetry. Figure 2.8(a) shows an original image from the bench vise sequence, where the bench vise is placed in the middle and the background is cluttered with other objects. Figure 2.8(b) shows the overlay of the bench vise object model in the annotated 6D pose. It can be observed that the bench vise is not occluded and that the main challenge is the textureless object appearance and the background clutter. Figure 2.9(a) shows the meshed models for 13 out of the 15 objects.

2.4.2 Occluded LineMOD Dataset

One challenge missing from the LineMOD dataset is occlusion. To fill in this gap, Brachmann et al. annotated the 6D pose for the background objects in the bench vise sequence to introduce the occlusion challenge [14]. This generated the occluded LineMOD dataset, which contains 8 objects from the LM dataset. Figure 2.8(c) shows the overlay of the background object models in the corresponding annotated 6D pose. Compared to LineMOD, the occluded LineMOD dataset is more challenging.

2.4.3 YCB Video Dataset

The YCB video dataset is one of the largest datasets for benchmarking 6D object poses, and it was proposed by Xiang et al. [139] in 2018. This dataset contains 21 everyday life objects, namely: master chef can, cracker box, sugar box, tomato soup can, mustard bottle, tuna fish can, pudding box, gelatin box, potted meat can, banana, pitcher base, bleach cleanser, bowl, mug, power drill, wood block, scissors, large marker, large clamp, extra large clamp, and foam brick. Those 21 objects are

(a) (b) (c)

Figure 2.8: Examples of LM and LMO dataset. The pose annotation is visualized by transforming the target object model with the ground truth pose, and overlaying it on the image. (a) An original image. (b) The pose annotation of the target object in LM. (c) The pose annotations of the target objects in LMO.

(a) Objects in the LineMOD dataset.

(b) Objects in the YCB video dataset.

Figure 2.9: Examples of objects in the LineMOD (a) and the YCB video dataset (b).

selected from the YCB object set [19]. It contains 92 video sequences with a total of 133,827 frames. The 3D model of each object is provided in the point cloud format. The official train/test split uses 80 video sequences for training. Testing is performed on the 2,949 keyframes chosen from the remaining 12 sequences. 80,000 frames of synthetic data are also provided by the YCBV dataset as an extension to the training set. Figure 2.9(b) shows the objects models. The majority of the objects are low textured and have rotational symmetry.

2.4.4 Discussion

Apart from the objects and dataset size, one major difference between the YCB video dataset and the LineMOD dataset is the selection of test data. As mentioned before, the train and test data are selected from the same video sequences in LM. This means that the illumination conditions and object appearances are similar in the train and test data. Figure 2.10 shows a comparison between a train and a test image in LM. It can be observed that the illumination conditions are very similar. This means that this dataset is relatively easy for color-based methods.

In contrast, the train and test data of YCBV are selected from different video sequences. Figure 2.11 shows examples of train and test data of this dataset. It can be seen that the illumination condition and object appearance are very different. This implies that this dataset is potentially more challenging, especially for color-based methods.

From the perspective of depth data quality, the depth data quality is as good as expected in YCBV. On the other hand, the depth data is inaccurate in both LM and LMO. This is because the intrinsic parameters are inaccurate and the resulting pinhole camera model cannot provide an accurate depth estimation [68, 119]. This inaccurate depth information could potentially make the LM and LMO more challenging for depth-based methods, compared to YCBV.

In this thesis, we will present a point cloud-based data synthesis pipeline that generates training data in the point cloud representation. Our synthetic data is a good complementary to existing datasets (see Section 3.3.2 for detailed discussion). Our data synthesis pipeline is presented in Chapter 6.

2.5 Evaluation Metrics

This section answers the question of how we evaluate whether a 6D pose estimate is accurate or not. This is conducted by comparing the estimated pose and the ground truth pose with specific measures. There have been measures such as “5° and 5 cm” used for evaluating 3D rotation and 3D translation separately [52]. This particular metric means that a rotation estimate is considered correct if the rotation error is less than 5°, despite the direction of the error. Similarly, a translation estimation is considered correct if the translation error is less than 5 cm. This metric is relatively easy, but it contains two values and those two values might show conflicting results. Hinterstoisser et al. [52] proposed a pair of metrics that

(a) Train. (b) Test.

Figure 2.10: An example of train and test example in LM. The target object is the ape in the middle.

(a) Train. (b) Test.

Figure 2.11: An example of train and test example in YCBV. The target object is the power drill.

evaluate rotation and translation simultaneously. Those two metrics use the 3D object models. The first is called the average distance (ADD) of model points. Given a 3D model represented as a set P^O with n points $\mathbf{x} \in P^O$, ground truth rotation R and translation $\mathbf{t}$, as well as estimated rotation $\hat{R}$, and translation $\hat{\mathbf{t}}$, the ADD is defined as:

$$\text{ADD} = \frac{1}{n} \sum_{\mathbf{x} \in P^O} \left\| (R\mathbf{x} + \mathbf{t}) - (\hat{R}\mathbf{x} + \hat{\mathbf{t}}) \right\|_2 . \tag{2.16}$$

The other is called average distance for a rotationally symmetric object (ADD-S). ADD-S is computed using the closest point distance. It provides a distance measure that considers possible pose ambiguities caused by rotational symmetry:

$$\text{ADD-S} = \frac{1}{n} \sum_{\mathbf{x}_1 \in P^O} \min_{\mathbf{x}_2 \in P^O} \left\| (R\mathbf{x}_1 + \mathbf{t}) - (\hat{R}\mathbf{x}_2 + \hat{\mathbf{t}}) \right\|_2 . \tag{2.17}$$

A 6D pose estimate is considered correct if ADD and ADD-S are smaller than a given threshold. Typically, the area under the error threshold accuracy curve (AUC) for ADD and ADD-S is calculated with a distance threshold. For YCBV, a 6D pose estimate is considered correct if ADD and ADD-S are smaller than a given threshold of 0.1 m. This means the AUC for ADD and ADD-S is calculated with a maximum threshold of 0.1 m. For LM and LMO, a pose is considered correct if the error is less than 10% of the maximum diameter of the target object [52].

Chapter 3

State of the Art in 6D Pose Estimation of Known Objects

6D pose estimation is a well-studied topic in the computer vision literature since the early 2000s. In general, it can be divided into instance-level and category-level object pose estimation. The instance-level pose estimation assumes the target objects are known, for example, a particular mug or a particular power drill. The category-level pose estimation assumes the target objects are from specific known categories, but the exact object models are unknown. For example, if the known categories contain the category "mug", the system is expected to deal with mugs with various shapes, sizes, and textures. The majority of the research focuses on the instance-level object pose estimation during the past decade, and category-level object pose estimation is receiving more and more attention recently. This thesis focuses on the classic instance-level object pose estimation and deals with known target objects.

Some pioneer works were able to use sparse features detected in RGB images to give reliable object pose estimation [84]. However, those target objects are mostly well-textured, and the approaches proposed during this period make good use of the textures to develop fast and scalable methods. With the commercially available Kinect depth sensors in 2010, researchers started investigating this problem using RGB-D images. With the additional depth channel, the pose estimation of textureless objects was enabled [50]. Since then, researchers focused on challenges such as clutter [52], occlusion [14], and object symmetry [55] from RGB-D images. Meanwhile, some also try to achieve the same performance with only RGB images [90]. Moreover, with the rise of deep learning from other visual recognition tasks, many deep learning-based approaches also focus on achieving state-of-the-art performance with RGB images [68]. Apart from RGB or RGB-D representations, point clouds have also been used to extract features for object pose estimation [25].

In this chapter, we categorize the existing approaches according to the methods they use for inferring the 6D pose. In general, there are multi-stage methods and single-stage methods. Multi-stage methods normally contain multiple processing stages in their pipelines, and their pipelines often contain non-deep learning components. Single-stage methods refer to deep learning-based methods and they

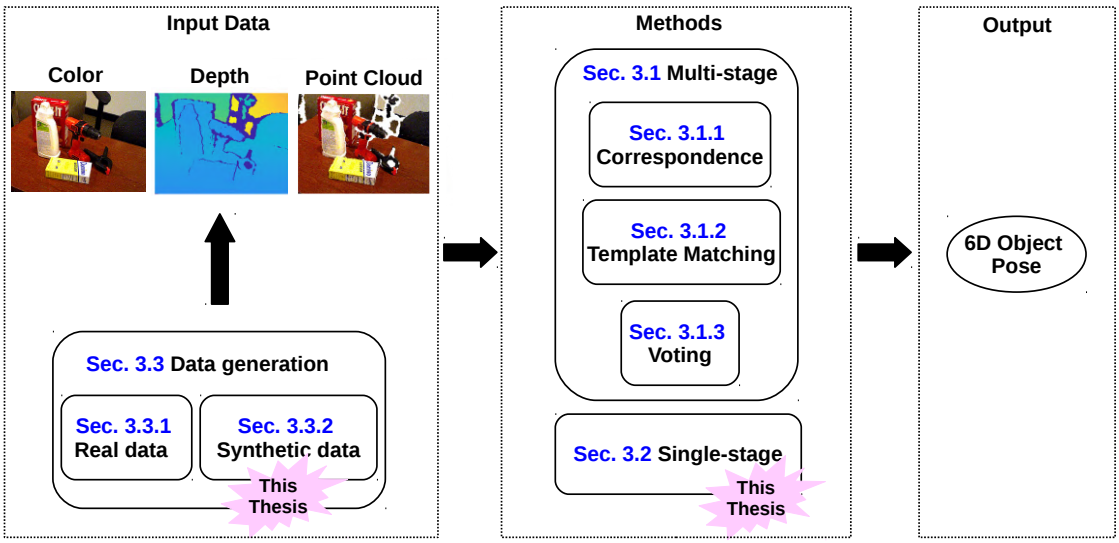


Figure 3.1: Overview of the literature chapter.

are typically trained end-to-end. We refer to them as the single-stage approaches in this thesis. Multi-stage approaches extract intermediate features or templates from the input images for obtaining the 6D pose estimate. Single-stage approaches directly infer the 6D pose from the input images. Previously, the multi-stage approaches were very dominating in the literature. With the rise of deep learning, some competitive single-stage approaches were also proposed.

Multi-stage methods based on 2D-3D or 3D-3D correspondence are discussed in Section 3.1.1. Multi-stage methods based on template matching are discussed in Section 3.1.2, and methods based on voting are presented in Section 3.1.3. The single-stage approaches are discussed in Section 3.2. Figure 3.1 gives an overview of the related work. The work presented in this thesis belongs to the single-stage category. Within each category, we discuss how the key aspects of each stream of methods have evolved and what their advantages or disadvantages are. On top of reviewing the methods, we mostly follow a chronological order when introducing the works to reveal the history of this research field.

Apart from the related work in estimating 6D poses, we also provide a review of how the training data is generated for 6D pose estimation. This is a relevant topic in this thesis because we are interested in integrating pose estimation systems into robotic applications and may encounter novel objects or pose distribution. In this case, it requires to collect or generate new training data. In this thesis, we also present a point cloud-based data generation pipeline. We review the existing data generation methods in Section 3.3.

3.1 Multi-stage Approaches

For multi-stage approaches, the popular representations are sparse features, dense features, and holistic templates. From those intermediate feature representations,

different methods such as correspondence, template matching, or voting are used to obtain the final 6D object pose.

3.1.1 Correspondence-based Approaches

Correspondence-based methods often have three stages: Firstly, some key features are extracted from the input data. The key features can be sparse features such as local image patches or dense per pixel features. Secondly, based on the features, the correspondences between the input image and the object model are established, and a pool of pose hypotheses is generated. Finally, with some similarity measures and geometric constraints, they search for an optimized hypothesis that best aligns the correspondences. Figure 3.2 shows an illustration of the multi-stage correspondence-based approach using sparse features as an example.

Sparse feature-based. Early methods for 3D object pose estimation rely on simple image features such as edges and corners. For example, a pioneer work [47] describes the target object as a set of shape primitives such as lines and matches the lines with the edge contours detected in the image to find a good 3D pose estimate. However, edge-based approaches are not robust and reliable in practice [78], as they cannot handle partial occlusions and background clutter can easily mislead the matching process. Comparing with using only contours, Lowe [84] proposes a much richer descriptor known as the scale-invariant feature transform descriptor (SIFT). A staged filtering approach is used to detect stable points in the scale space and forms SIFT keys. Those SIFT keys are local feature coordinates that are invariant to translation, rotation, scale, etc. With pre-defined training images, the SIFT features can be used for the nearest neighbor search to identify objects and their poses. Following this concept, faster descriptors such as SURF [8] and ORB [107] features were also proposed. Since those descriptors are discriminative local features robust to the change of viewpoint and illumination, they can be used for complicated scenarios [20].

Since only a small number of correspondences are required for recovering a 6D pose, those methods are generally robust to occlusion. Although it is very impressive that those local descriptors can handle very complicated real-life scenarios, it relies on a strong prior assumption that all the target objects are textured. Since this is not always the case with many objects in real life, research focus in this field has shifted to textureless objects.

Dense feature-based. For textureless objects, sparse features are not sufficient and error-prone, hence denser features are desired. Motivated by the work in the field of articulated human pose estimation, Brachmann et al. and Zach et al. [14, 147] adapt the idea of object coordinate representation for 6D object pose estimation. The hypothesis is that although the individual correspondences predicted for areas with less texture may be inaccurate, the large quantity of them can still contribute to an accurate pose estimate [13]. In the object coordinate repre-

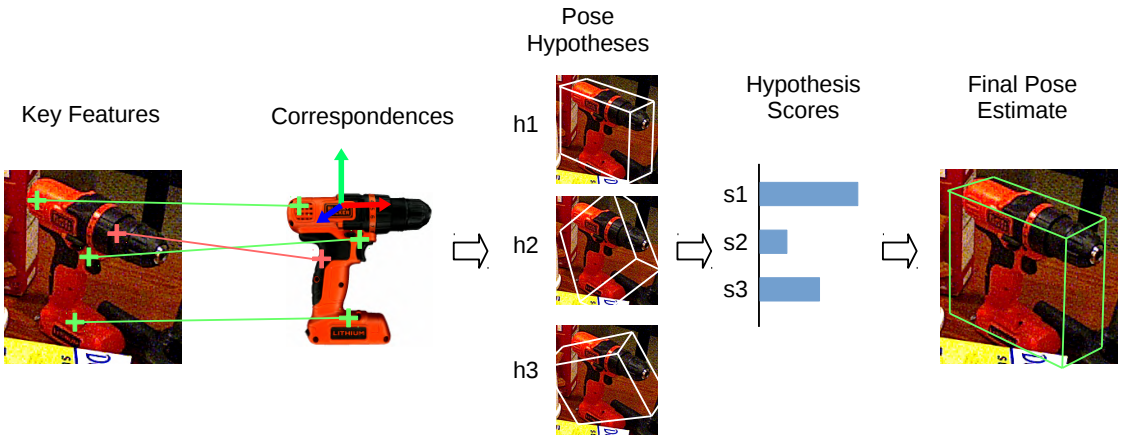


Figure 3.2: Correspondence based methods with sparse features [13]. The key features are extracted and used for prediction correspondences with the object model with an input image. Correct correspondences are denoted with green and false ones with red. A pool of pose hypotheses is generated from the correspondences. Some scoring functions are used to give scores to each hypothesis. The pose hypothesis with the best score is selected to be the final pose estimate.

sentation, each pixel in the image selects a coordinate on the object in a canonical pose [14]. Comparing with the aforementioned sparse feature, this per-pixel object coordinate feature is referred to as dense features.

To give a detailed example, Brachmann et al. [14] use a random forest to learn the object class and object coordinate for each pixel in an RGB-D image. This information together with the depth information is used to form an energy function. A RANSAC-based algorithm is used to sample pose hypotheses, and the final pose estimate is found by energy minimization. An interesting method is used for calculating the energy for each pose hypothesis. A synthetic depth image is rendered with an estimated pose and compared with observed depth values. This rendering and comparing approach is usually referred to as analysis-by-synthesis.

Building on the analysis-by-synthesis idea, the energy function is replaced by a CNN in [74]. This is one of the first works that uses deep learning in a 6D pose estimation system and it was proposed in 2015. Their motivation is that the energy function has only a few parameters, the CNN used in [74] has around 600K. The richness of parameters can potentially help the method to achieve better performance. By using a CNN, it shifted from designing how to compare to learning how to compare. The CNN is used as a probabilistic model and trained with a maximum likelihood objective [74]. Using this data-driven module for comparison, the performance of the pipeline is largely improved, which shows the potential of deep networks in this field. Recently, He et al. proposed a method to use deep networks for feature extraction from both color and depth information and use the extracted features for keypoints detection [49]. The keypoints are then used for least-squares fitting for finding the 6D pose.

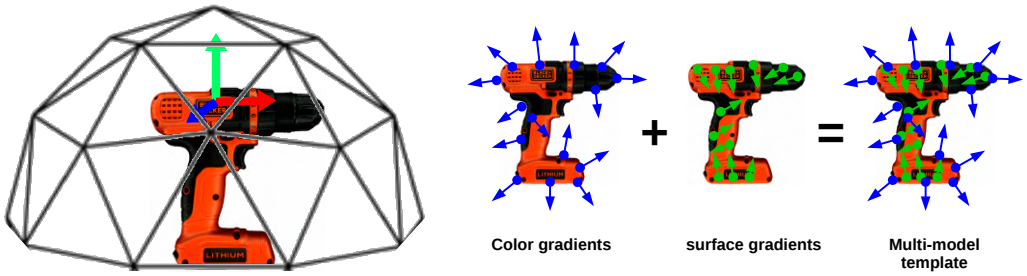
So far, all of the approaches are using RGB-D images, and the popular benchmarking dataset at the time (LineMOD) is getting closer to be "solved". To raise the bar, Brachmann et al. [15] propose a method that uses only RGB information to achieve good performance. Similar to [14], per pixel object coordinates and class labels are obtained via a random forest, followed by RANSAC pose sampling and refinement. The major difference to [14] is that the random forest is extended to an auto-context random forest, with which the uncertainties of the learned class labels and object coordinates are also considered in the iterative learning process. The uncertainty-driven technique boosts the method performance. This approach shows that it is also possible to predict accurate object poses with only RGB information. Up to the time of writing this thesis, one of the main research trends in this field was to attempt solving benchmarks with only RGB information.

3.1.2 Template Matching Based Method

In this section, we introduce an alternative representation to image features, which is called the template. The main difference between templates and image features is that a template is a global object representation while a feature is a local representation. The template can either be handcrafted or learned with deep networks.

Hinterstoisser et al. are the first to propose using templates for representing texture-less objects in 2011 [50, 51, 52]. The main idea is to use the image gradient as the templates. The image gradient contains color and depth gradient. Moreover, the image gradient is proven to be more discriminant than descriptors such as SIFT and robust to illumination changes [50]. Apart from color images, the gradient from depth information is also extracted to create a multi-modal template. Figure 3.3(a) is an example of the template collection setup. Given a 3D object model, the templates are collected at fixed distances and cover the half-sphere of the viewpoints. Approximately 2000 templates are needed for detecting an object. Figure 3.3(b) shows an example of the multimodal template. For pose estimation, similarity measures are applied to calculate the similarity between a reference template and an input image. The detection results are further improved by adding additional tests using color and depth information to reject outliers [52].

One recent development in the deep learning domain for template-based object pose estimation is the augmented autoencoder (AAE) [119]. Proposed by Sundermeyer in 2018, the AAE is used for 3D rotation estimation and it is a variant of the Denoising Autoencoder [128]. For estimating a 6D object pose, the authors first use 2D bounding boxes for translation estimation and then use the AAE for 3D rotation estimation. The key idea of the AAE is to control which properties are encoded by the latent representation and which ones are ignored [119]. Properties that shall be ignored are treated as data augmentation and applied to the original input images of the encoder. The decoder is then trained to produce an output without those augmentations. For example, if the AAE shall be color invariant, all kinds of color augmentation should be applied to the original input image. The task of the decoder is to reconstruct the original input image without any color augmentation. On the other hand, the properties that shall be encoded are controlled



(a) (b)

Figure 3.3: (a) Setup for template collection. The vertices represent the virtual camera centers used for template generation. (b) An example of multi-modal template.

by the reconstructed target of the decoder. In [119], the decoder reconstructs an image that contains the target object in the same 3D rotation as the input image. In this way, the latent representation encodes the 3D rotation information of the input image, hence it can be used as a template representing the 3D pose of the input. In other words, rather than explicitly mapping from an input image to a pose label, the AAE learns the implicit representation of object orientation in a latent space. A codebook of latent representation is created off-line, and they use the nearest neighbor search to compare a test representation within the codebook.

For the handcrafted template, since it uses the whole object image as the template, those approaches are also referred to as holistic methods in the literature. Comparing with the early approaches mentioned above, this holistic template matching scheme is much more robust in cluttered scenes for texture-less objects. However, due to their holistic nature, the templates are not robust to occlusion. Nonetheless, this holistic template inspired researchers to adapt it for a more occlusion resistant representation. The deep-learned templates can be robust towards occlusion if the training data was augmented with occlusion. Due to the flexibility of data augmentation, it has a lot of potential to be extended. For example, the AAE is improved by [134] by adding edge priors.

3.1.3 Voting Based Methods

As mentioned above, holistic templates are not robust to occlusion. To deal with this shortcoming, researchers propose to adapt it to a local patch descriptor. Instead of comparing the templates or looking for correspondences for estimating poses, those local patch descriptors are used in a voting scheme. Tejan et al. [121] first adapt the multimodal template idea and create scale-invariant local patch features. The scale-invariance is achieved by using the depth of the patch center to scale the offset. These local patch features are integrated into a random forest framework and used for voting 6D object poses. They have shown that using local patch features helps to handle occlusions.

However, since templates are hand-crafted features, it is challenging to make them discriminative for a large range of everyday objects [24]. Hence, more scalable

approaches are desired. Therefore, Doumanoglou et al. [24] propose to learn the local patch features in an unsupervised manner with deep sparse autoencoders. Moreover, instead of using similarity measures for voting, the learned features are fed to a Hough Forest, and 6D Hough voting is used to determine object class and 6D poses. Kehl et al. [69] propose a similar approach where they use convolutional autoencoder (CAE) to learn local RGB-D patches and a k-nearest neighbor (k-NN) search for vote casting.

Above mentioned methods all rely on RGB-D images. There are also works focusing on estimating object poses from depth information represented by point clouds. One of the pioneering works on pose estimation with point clouds is from Drost et al. [25] in 2000. They propose a global descriptor based on oriented point pair features (PPF). During testing, point pair features are extracted locally and matched to the model descriptor. The matches are used in a fast voting scheme for finding an optimal object pose. This method is further improved by Hinterstoisser et al. using a novel sampling and voting scheme [53]. It is worth mentioning that this point cloud-based feature is still used by the winning method in a recent benchmark challenge in 2019 [58].

3.2 Single-stage Approaches

The methods mentioned above are multi-stage methods, and their pipelines contain non-deep learning approaches. In recent years, methods that rely mostly on deep learning components have been proposed. Because they are typically trained end-to-end, we refer to them as the single-stage approaches. Approaches in this category can be divided into regression-based methods and classification based methods. Regression-based methods assume their output is in a continuous space and directly regresses to the desired values. The regress target can either be 6D object poses or the image coordinate of 3D bounding boxes. Classification based methods assume their output space is discrete and treat the 6D pose estimation as a classification task.

One example of the regression-based methods is PoseCNN proposed by Xiang et al. [139]. Using RGB images and CNNs, they regress to object centers for translation estimation and quaternion values for rotation prediction. It is worth mentioning that they further refine the pose with iterative closest point (ICP) using depth information. This is a common way to combine forces from color and depth information. Wang et al. propose a different way to combine color and depth information [130] in a system named DenseFusion. They use CNNs to extract features from RGB images and PointNet to extract features from point cloud segments. Those features are combined for translation and rotation regression.

Kehl et al. propose SSD-6D to "make RGB-based 6D object pose estimation great again" with CNNs in a classification manner [68]. They extend the popular single shot detector (SSD) paradigm [83] and classify the sampled viewpoints and in-plane rotations of target objects. Similarly, Qi et al. also formulate rotation estimation as classification and discretize the rotation angles to bins [102]. They

use a variant of PointNet for rotation classification on point cloud segments. Li et al. also propose a classification-based approach that extracts features from RGB images and depth information separately [80]. They tried out an interesting idea to convert the depth map to a point cloud and format the point cloud as a matrix with 3D coordinates as 3 image channels named XYZ map. This XYZ map was then processed with CNNs.

It can be seen that existing approaches mainly exploit color information [139, 68], or combine color and depth information [80]. The main difference between the approaches presented in this thesis and those existing ones is that we mainly exploit depth information. Regarding the depth-based method by Qi et al. [102], they only report experimental data for pose estimation for a single angle as opposed to all three as we do. Furthermore, they formulate rotation estimation as classification and discretize the rotation angles to bins. It is not clear if this approach would scale up to three rotation angles. In contrast, our methods predict the full 6D object pose, including the 3D rotation. The methods presented in this thesis fill the void of what depth information can contribute to 6D object pose estimation.

Although the single-stage approaches are not currently the best performing methods on benchmarks, these single-stage approaches gained popularity due to their end-to-end nature and simplicity. Overall, it is still intriguing to explore and improve the performance of single-stage methods.

3.3 Data Generation

In this section, we review existing data generation methods. There are two ways of obtaining training data, and one is to collect real images of target objects and annotate the 6D object poses. Another way is to generate synthetic images using target object models, and there is no need for manual pose annotation. The advantage of collecting real training data is that the training data is visually similar to real-world testing data, and the disadvantage is that 6D object pose annotation is difficult. Generating synthetic data omits the need for pose annotation. However, there is often a difference in appearance between real and synthetic data, it is called the visual reality gap. Hence, some effort is required to bridge this gap and make synthetic images visually similar to real-world testing images.

3.3.1 Real Data

Although challenging, there are ways to annotate 6D object poses in real images. Marion et al. proposed a pipeline for generating RGB-D data with pixel-precise labels and 6D object poses [88]. They collect a sequence of images of a scene and produce a 3D scene reconstruction. Then they label the object poses using a human-assisted ICP on the scene reconstruction. Finally, they obtain the pose labels for all RGB-D images by reprojecting the pose labels. Xiang et al. use a similar approach to annotate the YCB Video Dataset [139]. They manually label

the object poses in the first frame and use the camera trajectory to find the relative object poses in other frames.

3.3.2 Synthetic Data

Although there are ways to manually label poses, researchers attempt to invest less effort in data annotation and look into generating synthetic data. To create synthetic data, the viewpoint and the object's appearance from that viewpoint must be determined. Object appearance can be obtained from either textured 3D object models or real-world training datasets. Viewpoints are selected using some simple or dataset-based heuristics.

Hinterstoisser et al. propose an approach that defines spheres around the textured 3D object model with fixed radii and sample viewpoints from a hemisphere that covers the upper part of the object mode [52]. The object model is then rendered in the target viewpoint onto the plain or random background. This simple approach is also used in other works [149, 119, 134]. The advantage of this method is that it provides good coverage for 3D rotation. The limitation is that using fixed radii puts constraints on the coverage of 3D translation. This approach can be extended to render object models in arbitrary poses on a random image [123]. Figure 3.4(a) shows an example.

Another way to obtain the object's appearance and viewpoint is by segmenting foreground objects from an existing training set [122, 105, 98]. This is also known as the "cut&paste" approach, which works well for tasks such as 2D object detection and instance segmentation [59]. More data augmentation, such as background noise, can be applied to the rendered image for reducing the visual reality gap [123]. However, due to the visual reality gap, this strategy tends to be insufficient by itself for the 6D object pose estimation task and causes performance drop [57]. Moreover, this approach also relies on having the annotated real training data. Figure 3.4(b) shows an example image.

Some approaches synthesize data with photo-realistic appearance and physically plausible object poses [123, 59, 91, 23]. This approach requires a render to create object appearances with realistic lighting and reflection and physics simulators to provide physically plausible poses. Rendering is computationally expensive, and the data created off-line requires large amounts of hardware storage. Figure 3.4(c) shows an example image of a photo-realistic image.

Comparing with existing methods, the main difference is that ours is a depth-based data generation pipeline. Although methods such as BlenderProc [23] also provides synthetic depth map as a byproduct, the main focus of such method is the color information. Overall, existing methods focus on using textured object models or segments to generate color-based synthetic data, and there are no depth-based generation methods. Hence, this thesis fills this gap.

(a) Rendered object models onto a random background. Adapted from [123].

(b) Cut & paste object segments onto a random background. Adapted from [122].

(c) Photo realistic rendering. Adapted from [59].

Figure 3.4: Examples of synthetic data. Synthetic images created by rendering object models (left), cut & paste existing object segments (middle), and photo realistic rendering (right).

3.4 Discussion

The research field started with relatively simple scenarios with textured objects, and the researchers showed that this task can be handled with hand-crafted features [84]. The bar is then raised to texture-less objects, and it is still the main focus to this day [50]. Data modality wise, the commonly used modalities are color and depth [50, 130]. At the moment, one main trend is using only color information to achieve good results on benchmark datasets [68], and the other is using both modalities to achieve even better results [130]. Methodology wise, the current trend is to combine both non-deep learning and deep learning approaches in a multi-stage pipeline and attempt to use the best of both worlds [139]. On the other hand, the single-stage paradigm is still attractive due to its simplicity and competitive performance [68].

Having been created in 2012 and 2014, LineMOD and Occluded LineMOD dataset became the most popular benchmarks for 6D pose estimation. Researchers started with solving those datasets with color and depth data [14, 74]. As a result, the test data in LineMOD is almost solved, and the state-of-the-art performance on the challenging Occluded LineMOD is approaching 80%. To increase the challenge and make use of the popular CNN, the research focus shifted from RGB-D data to RGB only. With the larger and more challenging YCB video dataset coming into the game, researchers start to bring back the depth modality into a deep learning framework [139, 130, 49].

There is hardly any work investigating what depth information can solely contribute to 6D pose estimation in a deep learning framework. We focus on filling this missing piece in this field. Furthermore, we will evaluate our methods on all datasets mentioned above for more insights. Methodology wise, we will follow the trend and attempt to integrate both traditional and deep learning methods in the same pipeline for good performance.

Chapter 4

CloudPose: Learning 6D Object Pose Regression on Point Clouds

In the previous chapter, we introduced the state-of-the-art deep learning-based methods for estimating 6D object pose. All of those methods rely on either color [68, 94, 123] or color and depth information [139, 130]. Inspired by the success of CNNs in other visual recognition tasks, it is common to learn deep features from color images and use the features for object pose inference. On the other hand, when depth information is considered, it is usually treated as an additional channel to the RGB image and processed with CNNs [69, 148, 80, 17]. We also introduced some methods that use depth information represented by point clouds for 6D object pose estimation. However, in existing methods, point clouds are used for geometry based pose refinement [109, 111, 139, 63] or template matching with hand-crafted point cloud features [25, 53].

In general, comparing to color information, depth information is usually regarded as auxiliary information. However, depth contains rich geometric information of the object shape, and intuitively it is very useful for inferring the object pose. Current approaches that use CNNs to process depth information require the unnatural matrix structure of depth information and can not be applied to unstructured depth data (e.g., from a laser range finder). On the other hand, deep-learned depth features should be used since they are usually more effective than their hand-crafted counterpart. Moreover, existing methods tend to use point clouds only in the registration stage, which confines its usage scope.

Basing on those observations, we investigate the task of estimating the 6D object pose from depth information represented by point clouds in this chapter. We present a supervised learning framework that infers 6D object poses from point clouds. The point clouds are used as input for both deep networks and geometry-based pose refinement. Since this was the first framework that directly regresses to 6D object poses from depth information, we consider four fundamental aspects for designing the system. Moreover, inspired by the success of PointNet [103] in classification and segmentation tasks, we decided to use a PointNet like structure as the basic building block.

The four aspects for system design are presented in Section 4.1. Based on the design choices, we present the system architecture in Section 4.2. The formulation of loss functions for 6D pose regression is described in Section 4.3. We describe the pose refinement method in Section 4.4. We show the effectiveness of the proposed system with experimental results in Section 4.5. The evaluation of each system component is also provided in this section. Some system insights are also presented. Finally, we summarize this chapter with discussion and insights on the proposed system in Section 4.6.

4.1 Design Choices

For designing a supervised learning system for 6D object regression on point clouds, we consider four aspects. The first aspect is: "what is a suitable representation for depth information in a deep learning-based system?". As mentioned before, treating depth as an additional channel to RGB information and using CNNs for feature extraction is useful for specific tasks. However, color and depth are two inherently different data modalities, so it is unclear whether it is an efficient way to treat depth information as a structured matrix. A depth map only contains the distance information along one (z-axis) out of three axes (x, y, z-axis). In contrast, its corresponding point cloud contains the full 3D information. To this end, we argue that the point cloud representation provides more useful information for a deep learning system. Furthermore, to exploit the point cloud structure, it should be used in the scope of both deep networks and geometry-based optimization.

The second aspect is whether translation and orientation should be estimated with separate networks or a single network in a supervised learning system. During supervised learning, a network learns the mapping from its input to the desired output guided by a loss function. If one network shall learn to map to a 6D pose, it needs to cope with rotation and translation losses simultaneously. Since the metric units for translation and orientation are different (i.e., meters and radius), it is potentially very challenging for one network to cope with two inherently different metrics. We hypothesize that regressing them using separate networks and loss functions is a more suitable choice.

The third aspect is the choice of rotation representation. Due to its popularity in robotic applications, quaternions have been a popular choice for many learning-based systems [139, 130]. However, as introduced in Section 2.2, quaternions have the unit-norm constraint. While this constraint may not impact non-learning methods, it imposes a limit on the network output range. This means a range of the network output is not a valid rotation representation and makes the learning task more complicated. On the other hand, the axis-angle representation is constraint-free, and we argue that it is a more suitable choice for a learning-based method.

The fourth aspect is choosing a suitable loss function for measuring the distance between two rotations. Euclidean distance (L_2 distance) is a popular choice for measuring the distance between two rotations [130, 70]. However, comparing to the L_2 distance, geodesic distance measures the distance between two rotations.

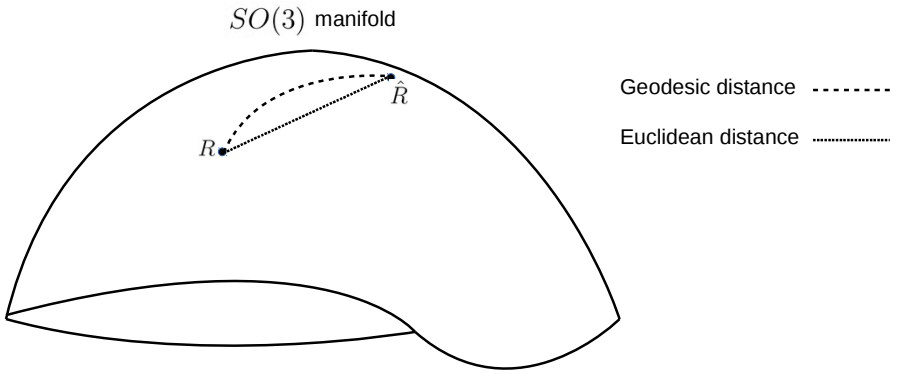


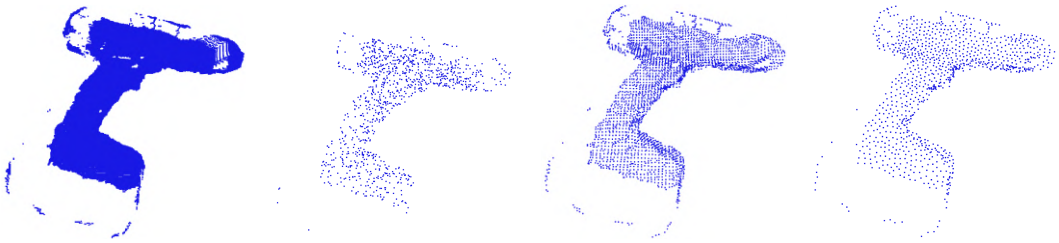
Figure 4.1: Illustration of geodesic distance and Euclidean distance between two rotations R and $\hat{R}$ in $SO(3)$.

Figure 4.1 shows an illustration of geodesic and Euclidean distance for two rotations R and $\hat{R}$ in $SO(3)$. $SO(3)$ stands for the special orthogonal group with 3 dimensions, and it is a group that contains all the 3D rotations. Details regarding $SO(3)$ can be found in A.2. We argue that the geodesic distance is a more suitable choice because it is more mathematically justified.

4.2 System Architecture

After considering the design choices, we propose a multi-class system for object 6D pose estimation. A multi-class system means that we can use the same system to predict poses for objects of different classes. The required input to the pose estimation networks is an object segment and its corresponding class information. For obtaining the object segment and its class information, semantic segmentation can be used. As RGB-based semantic segmentation is a well-studied problem and it is not in the scope of this thesis, we assume the object segment and class information is provided by an off-the-shelf method and focus on the object pose estimation from a point cloud segment. Here, we use the semantic segmentation from [139].

The original segment obtained from the semantic segmentation normally contains a large number of points, and it is computationally expensive to process. Therefore, it should be downsampled before further processing. There are several ways to downsample a point cloud segment, such as random downsampling, voxel downsampling, and Farthest Point Sampling [27]. Figure 4.2 shows an original segment and its downsampled version with random downsample, voxel downsample, and FPS, respectively. Among those methods, the random down sample can not ensure even coverage of the object surface, and voxel down sampling introduces quantization artifacts. In contrast, with more than 30 times fewer points, the FPS segment preserves the surface structure of the original segment with more consistency. To ensure a consistent surface structure representation, we process the segment with FPS.



(a) Original segment with 36396 points

(b) Randomly down sampled segment with 1024 points

(c) Voxel down sampled segment with 2805 points

(d) FPS segment with 1024 points

Figure 4.2: A original segment and a down sampled segment.

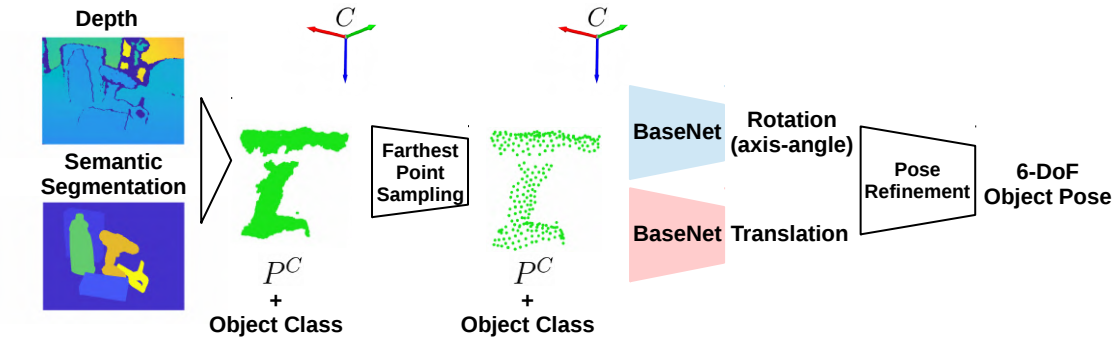


Figure 4.3: System overview. A point cloud P^C is created using the depth data and the output from a semantic segmentation method. This segment is processed with farthest point sampling to obtain a downsampled segment with a consistent surface structure. The segment with object class information is fed into two networks for rotation and translation prediction. BaseNet is a variant of PointNet. The geometry-based iterative closest point algorithm is used for pose refinement.

An overview of the system is shown in Figure 4.3. From a depth map and its corresponding semantic segmentation, we can obtain the target object segment from the depth map. Assuming the intrinsic camera parameters are known, we can calculate the corresponding point cloud segment P^C in the camera coordinate system. The resulting point cloud segment is downsampled to create a segment with fewer points. Since they represent the same object surface, we also use P^C to present the downsampled segment for notational simplicity. This downsampled segment and class information are combined as the input for two separate networks for rotation estimation in axis-angle representation and translation prediction through translation residual regression. The 6D pose is refined with a geometry-based optimization process to produce the final pose estimate.

The BaseNet in the overview (Figure 4.3) is the basic building block of our system. BaseNet is an adapted version of PointNet [103], and Figure 4.4 shows its detailed structure. Assuming the total number of object classes is k, we convert the class labels into a one-hot encoding vector with size k. Given a point cloud segment with n points as input, we concatenate the one-hot encoding class information to

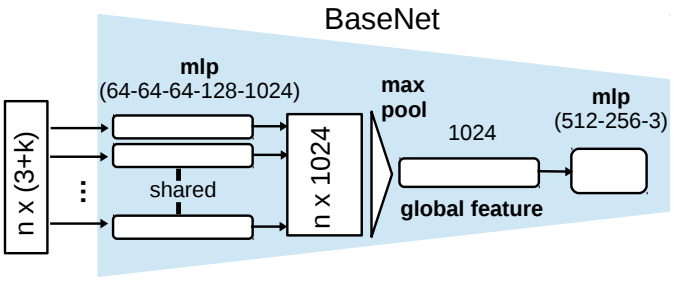


Figure 4.4: The architecture of BaseNet. The numbers in parentheses are numbers of MLP layers. Numbers not in parentheses indicate the dimensions of intermediate feature vectors. A feature vector for each point is learned with shared weights. A max-pooling layer aggregates the individual features into a global feature vector. A regression network with 3 fully-connected layers outputs either the translation or the rotation.

each point. This results in an input matrix with dimension $n \times (k+3)$, where 3 comes from the 3D (x, y, z) coordinates of each point. Each point with class information is processed independently using a series of multi-layer perceptrons (MLPs) with shared weights ($(64 - 64 - 64 - 128 - 1024)$ in Figure 4.4). A 1024-dimension feature vector is learned for each point (the $n \times 1024$ matrix in Figure 4.4). These feature vectors are max-pooled to create a 1024-dimension global feature vector, which is the global representation of the input point cloud. Finally, we use a three-layer regression MLP ($(512 - 256 - 3)$ in Figure 4.4) on top of this global feature to predict the poses. Since we are interested in predicting the 6D object pose, compared to PointNet, we remove the spatial transformer blocks.

With the building block BaseNet, Figure 4.5 shows a detailed diagram of our system. We use two separate networks to regressing translation and rotation, respectively. As described above, the input to both networks is the $n \times (3+k)$ matrix containing both the 3D point coordinate and its class information. The output of the rotation network is the estimated rotation in axis-angle representation.

Unlike the amount of rotation, which is in a fixed range between 0 and 2π, the amount of translation can be in various ranges depending on the data collection process. For example, the distance between the target object and the camera can be as near as 1 meter or as far as 5 meters. This is a large range space for tasks that require sub-centimeter or even sub-millimeter precision. In other words, the learning space for rotation network is fixed in the reasonable range of $[0, 2\pi)$ while the learning space for translation network can be much larger. To simplify the learning task, we adapt the normalization technique, which reduces the numeric data values into a small scale, without losing information [41]. To reduce the variance in learning space, we normalize the 3D point coordinates by removing its coordinate mean for the translation network. The translation network estimates the translation residual, and the full translation is obtained by adding back the coordinate mean. More details on this can be found in Section 4.3.

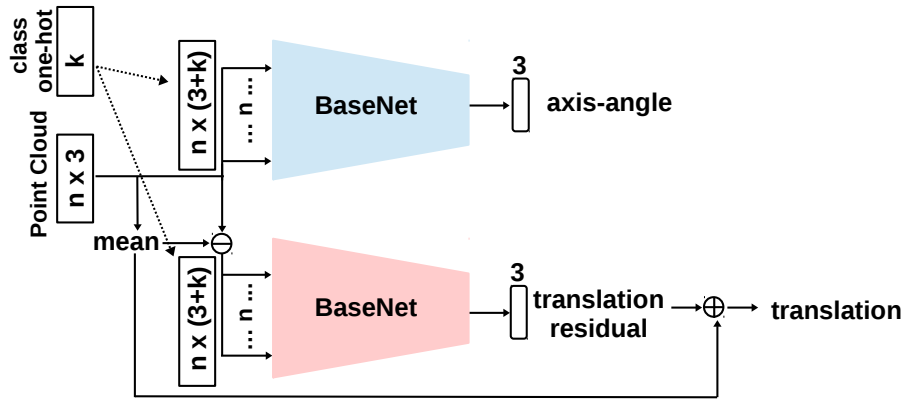


Figure 4.5: Diagram for input and output of our pose networks. For the rotation network, the input is point coordinate information concatenated with class information per point, and the output is rotation in axis-angle representation. For the translation network, the input coordinates are normalized by removing the mean. It outputs translation residual. The full translation is obtained by adding back the coordinate mean. The number of input points is n, and k is the total number of classes.

4.3 Loss Functions for 6D Pose Regression

We formulate the 6D pose estimation problem as a supervised learning problem, in which the network takes in a point cloud segment of a known object as the input and outputs the 6D pose of this segment. This section describes how the loss functions for 6D pose regression are formulated.

Recall from Section 2.1.1, given a known object represented by set $P^O = \left\{\mathbf{x}_i^O \in \mathbb{R}^3 \mid i = 1, \ldots, m\right\}$ and a set of points $P^C = \left\{\mathbf{x}_i^C \in \mathbb{R}^3 \mid i = 1, \ldots, n\right\}$ on the surface of this known object in the camera coordinate C, the aim of pose estimation is to find a transformation that transforms $\mathbf{x}_i^O$ from the object coordinate O to the camera coordinate C. This transformation is composed of a 3D translation $\mathbf{t}$ and a 3D rotation R.

For supervised learning, suitable loss functions are required to measure the differences between predicted poses and ground truth poses. In our case, we need to define the loss for translation and rotation separately. We use the axis-angle representation $\mathbf{r} \in \mathbb{R}^3$ as the learning target for rotation learning. Geodesic distance is used as the loss function for rotation regression. For translation learning, we predict the residual of translation.

Rotation estimation As shown in Section 2.2, in the axis-angle representation, a vector $\mathbf{r} \in \mathbb{R}^3$ represents a rotation of $\theta = \|\mathbf{r}\|_2$ radius around the unit vector $\frac{\mathbf{r}}{\|\mathbf{r}\|_2}$ [48]. Given an axis-angle representation $\mathbf{r} = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}^T$, the corresponding rotation matrix R is obtained via the exponential map exp :

$$R = \exp(\mathbf{r}_\times) = \mathrm{I}_{3\times 3} + \frac{\sin\theta}{\theta}\mathbf{r}_\times + \frac{1-\cos\theta}{\theta^2}\mathbf{r}_\times^2, \tag{4.1}$$

where $\mathrm{I}_{3\times3}$ is the identity matrix and $\mathbf{r}_\times$ is the skew-symmetric matrix

$$\mathbf{r}_\times = \begin{bmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{bmatrix}. \tag{4.2}$$

For rotation learning, we regress to a predicted rotation $\hat{\mathbf{r}} \in \mathbb{R}^3$. Prediction $\hat{\mathbf{r}}$ is compared with ground truth rotation $\mathbf{r}$ via a *rotation loss function* l_r, which is the geodesic distance between $\hat{R}$ and R [62, 48]:

$$l_r(\hat{\mathbf{r}}, \mathbf{r}) = \arccos\left(\frac{\mathrm{trace}(\hat{R}R^T) - 1}{2}\right), \tag{4.3}$$

where $\hat{R}$ and R are the two rotation matrices corresponding to $\hat{\mathbf{r}}$ and $\mathbf{r}$, respectively. This loss function directly measures the magnitude of rotation difference between $\hat{R}$ and R, so it is convenient to interpret. Furthermore, the network can make constraint-free predictions with axis-angle representations, in contrast to, e.g., quaternion representations, which require normalization.

Translation residual estimation As mentioned in Section 4.2, to reduce the variance in learning space, the input 3D point coordinates are normalized for the translation network. So, instead of learning the full translation, the network only needs to learn the displacement of the input coordinates with respect to the object center. In other words, the network learns the residual of translation, which is a simpler learning task. Figure 4.6 shows an illustration. The camera, object, and normalized coordinate frames are denoted with C, O and N, respectively. Assuming a target object (denoted with the grey square) is viewed by the camera from viewpoint C, and the red lines denote its visible part from the C, the full translation is the distance between O and C (denoted with blue and green dotted line). By removing the coordinate mean of the visible part, the object coordinate is moved to a normalized frame N (denoted with dotted gray and red lines). The translation residual is the distance between N and C (denoted with green dotted line).

Given a translation residual $\widehat{\Delta\mathbf{t}}$, the full translation prediction $\hat{\mathbf{t}}$ is obtained via

$$\hat{\mathbf{t}} = \widehat{\Delta\mathbf{t}} + \mu_{\mathbf{t}}, \tag{4.4}$$

where $\mu_{\mathbf{t}}$ is the mean of P^C. $L2$-norm is used to measure the distance between prediction $\hat{\mathbf{t}}$ and ground truth $\mathbf{t}$, resulting in the *translation loss function* $l_t(\hat{\mathbf{t}}, \mathbf{t})$:

$$l_t(\hat{\mathbf{t}}, \mathbf{t}) = \left\|\mathbf{t} - \hat{\mathbf{t}}\right\|_2. \tag{4.5}$$

Total loss function for 6D pose regression The total loss is defined as the combination of the translation and the rotation loss:

$$l(\mathbf{t}, \hat{\mathbf{t}}, \mathbf{r}, \hat{\mathbf{r}}) = \alpha l_t(\hat{\mathbf{t}}, \mathbf{t}) + l_r(\hat{\mathbf{r}}, \mathbf{r}), \tag{4.6}$$

where α is a scaling factor. The total loss is used for training the pose estimation networks.

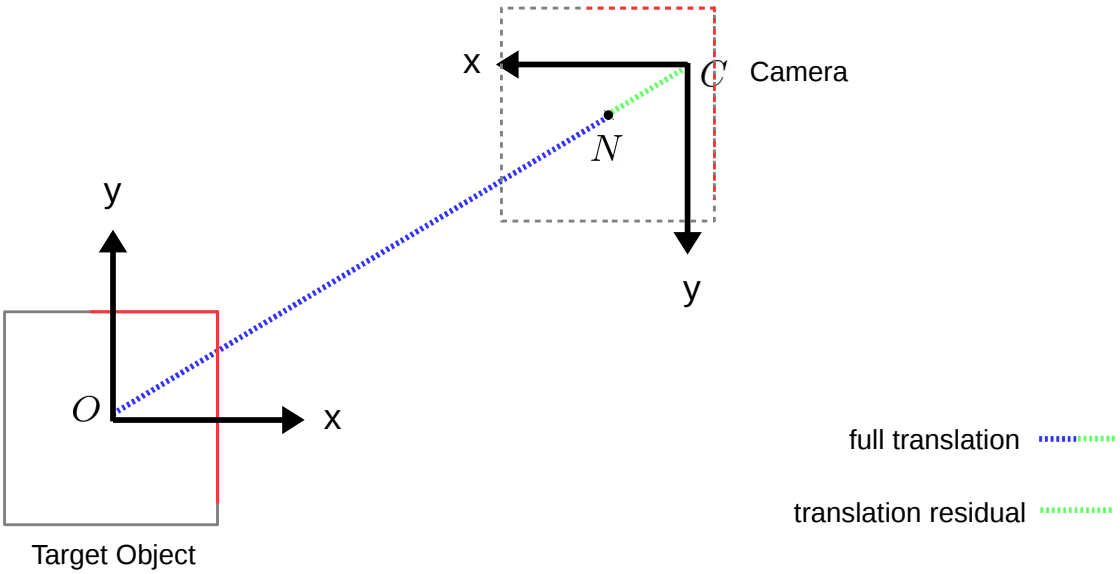


Figure 4.6: Illustration of translation residual. Camera, object, and normalized coordinate frame are denoted with C, O and N, respectively. The full translation is denoted with the blue and green dotted line, and the translation residual is in the green dotted line.

4.4 Iterative Closest Point

Iterative closest point is a method for minimizing the positional difference between two sets of point clouds. It is original proposed by Arun [3] and further improved by Besl and McKay [10]. There are several variant of ICP, here we introduce the vanilla point-to-point ICP that we use for pose refinement.

Assuming there are two point sets, the reference set $P = \{\mathbf{x}_i \in \mathbb{R}^3 \mid i = 1, \ldots, n\}$ and the source set $P' = \{\mathbf{x}'_i \in \mathbb{R}^3 \mid i = 1, \ldots, n\}$, and,

$$\mathbf{x}'_i = R\mathbf{x}_i + \mathbf{t} + \mathbf{n}_i, \tag{4.7}$$

where R is a 3D rotation, $\mathbf{t}$ is a 3D translation and $\mathbf{n}_i$ is a noise vector. The aim of ICP is to find R and $\mathbf{t}$ that minimize

$$\Sigma^2 = \sum_{i=1}^{n} \|\mathbf{x}'_i - (R\mathbf{x}_i + \mathbf{t})\|^2. \tag{4.8}$$

The first step is to find the nearest point in P for each point in P' with the distance d, which is

$$d(\mathbf{x}'_i, P) = \min_{j \in \{1,\ldots,n\}} d(\mathbf{x}'_i, \mathbf{x}_j). \tag{4.9}$$

To ensure plausible correspondence, the nearest point pair is removed if

$$d(\mathbf{x}'_i, \mathbf{x}_j) > \tau, \tag{4.10}$$

where τ is a predefined threshold.

Based on the remaining corresponding pairs, a rotation R' and a translation $\mathbf{t}'$ is found using Singular Value Decomposition (SVD). Then, P' is transformed to P'' with

$$\mathbf{x''}_i = \mathbf{R'x'}_i + \mathbf{t'} \tag{4.11}$$

and $P'' = \{\mathbf{x''}_i \in \mathbb{R}^3 \mid i = 1, \ldots, n\}$.

Now, P'' is the new source set and processes in Equations 4.9, 4.10, 4.11 are iterated as the the method's name suggested. The algorithm reaches convergence when the overall point distance between the source and the reference set is below a certain threshold.

4.5 Experiments

We evaluate the proposed system on the LineMOD, Occluded LineMOD, and YCB-Video datasets and compare the performance with the state-of-the-art methods PoseCNN [139] and DenseFusion [130]. We choose to compare with those two methods because they have the best performances on those datasets at the time of this work. Because YCB Video dataset was proposed by the authors of PoseCNN, methods earlier than PoseCNN such as SSD-6D [68] does not report on YCBV for comparison. We also compare the performance on a subset of the object classes with a state-of-the-art RGB-based method DOPE [123] on the YCBV dataset. Because the authors of DOPE only report their results on a subset of objects [123]. Besides the prediction accuracy (Section 4.5.2) and performance under occlusions (Section 4.5.3), we also investigate the impact of using different network structures, as well as the influence of different rotation representations (Section 4.5.4). We present insights regarding the number of points used by a trained system for rotation and translation regression (Section 4.5.5). The time performance is presented in Section 4.5.6.

4.5.1 Experiment Setup

During training, the Adam optimizer is used with a learning rate of 0.0008. The batch size is 128. For the total loss, we use $\alpha = 10$, which is given by the ratio between the expected error of translation and rotation at the end of the training [71]. The number of points of the input point cloud segment is $n = 256$. Batch normalization is applied to all layers. No dropout is used. For refinement, we use the Point-to-Point ICP registration provided by Open3D [151] and refine for 10 iterations. The initial search radius is 0.01m and is reduced by 10% after each iteration. For a fair comparison, all methods use object segmentation provided by PoseCNN during testing.

For YCBV, we follow the convention [139, 131] to train on the official training split as well as the provided $80,000$ synthetic data. All of the networks for YCBV are trained for 90 epochs. For LM, we also follow the official split, in which 15% of the dataset is used for training while 85% is used for testing. There is no official

	RGB	Depth	ICP	ADD	ADD-S	<1 cm
PC w/o ICP [139]	✓			51.5	75.6	26.1
PC [139]	✓	✓	✓	77.8	93.6	88.4
DF w/o ICP [130][1]	✓	✓		74.7	93.9	87.6
DF [130][2]	✓	✓	✓	76.3	**94.7**	89.0
Ours w/o ICP		✓		76.0	91.3	80.9
Ours		✓	✓	**82.7**	**94.7**	**90.3**

Table 4.1: Quantitative evaluation of 6D pose on the YCB-Video Dataset [139]. Best performance is in bold font.

synthetic training data. For LM, we generate synthetic validation data and train the network until we see convergence on the validation set.

4.5.2 Prediction Accuracy

YCB Video Dataset. We report the AUC score for ADD and ADD-S. The maximum thresholds for both curves are set to 0.1m. Furthermore, we also provide ADD-S accuracy with a threshold 0.01m (<1cm) to illustrate the performance accuracy under a smaller error tolerance. We first show the comparison of prediction accuracy of our method and the state-of-the-art methods. Evaluation results averaged for all 21 objects in the YCB-Video dataset are shown in Table 4.1. PoseCNN [139] uses RGB information to provide an initial pose estimate (PC w/o ICP), then uses depth information with a highly customized ICP for pose refinement (PC). DenseFusion [130] (DF w/o ICP) uses both color and point cloud features extracted by deep networks to give per-pixel pose estimate for final pose voting, and iterative pose refinement is performed with an extra network module. For a fair comparison, we also perform the ICP refinement on DF w/o ICP results (DF). Ours w/o ICP is the estimated pose from the proposed system architecture (Section 4.2), and Ours is the result after ICP refinement. For the overall performance in Table 4.1, we highlight the best performance in bold font. Details regarding the data type used by pose regression networks and the post process are also presented.

Our method achieves state-of-the-art performance using only depth information. In terms of ADD, we outperform both PC and DF. We observe that DF shows a small improvement compared to DF w/o ICP. One possible reason is the sensitivity of ICP to the initial pose guess, if the method already performs well without refinement, ICP can provide further gains. If the initial guess is poor, ICP can even make the results worse. This result indicates that features learned from depth information represented by unordered point clouds are sufficient for accurately regressing the 6D pose. Furthermore, this also shows that the proposed approach is an efficient way to use depth information in a deep learning framework for pose regression.

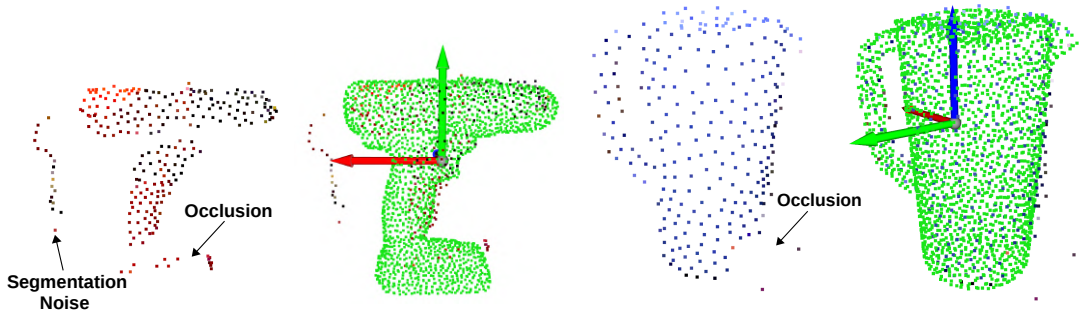
[1]This results are with iterative refinement.

	DOPE [123]		PC [139]			DF [130]			Ours		
	ADD-S	<1cm	ADD	ADD-S	<1cm	ADD	ADD-S	<1cm	ADD	ADD-S	<1cm
002_master_chef_can	—	—	68.1	95.8	99.5	**73.2**	**96.4**	100	46.9	95.4	95.4
003_cracker_box	62.7	29.6	83.4	92.7	84.8	**94.2**	**95.8**	97	76.7	93	80.4
004_sugar_box	85.0	33.4	97.1	98.2	100	96.5	97.6	100	**97.5**	**98.5**	99.7
005_tomato_soup_can	88.5	74.5	83.6	**96.6**	99	**87.4**	**96.6**	99.1	72.7	96.5	96.8
006_mustard_bottle	90.7	65.3	**98**	**98.6**	98.9	94.8	97.3	97.8	79.2	97.7	94.1
007_tuna_fish_can	—	—	**83.9**	97.1	97.6	81.8	97.1	99.5	72	**97.7**	100
008_pudding_box	—	—	**96.6**	**97.9**	100	93.2	95.9	98.6	94.4	97.3	91.1
009_gelatin_box	84.6	36.9	98.1	98.8	100	96.7	98	100	**98.6**	**99**	100
010_potted_meat_can	32.0	3.7	86	94.3	87.5	87.8	95	92	**90.6**	**95.7**	93.7
011_banana	—	—	91.9	97.1	95	83.6	96.2	98.2	**95.1**	**97.7**	95.5
019_pitcher_base	—	—	**96.9**	97.8	99.6	96.6	97.5	99.5	96.1	**97.9**	100
021_bleach_cleanser	—	—	92.5	96.9	95.1	89.7	95.8	99.4	**95.4**	**97.4**	98.4
024_bowl	—	—	14.4	81	42.9	5.9	89.5	55.7	**83.9**	**97.7**	99.3
025_mug	—	—	81.1	94.9	97.6	88.8	96.7	98	**93.9**	**97.8**	99.7
035_power_drill	—	—	**97.7**	**98.2**	99.3	93	96.1	97.8	94.9	97.7	96.7
036_wood_block	—	—	70.9	87.6	74.4	30.9	92.8	88.8	**90**	**94.9**	97.5
037_scissors	—	—	**78.4**	**91.7**	68	77.4	91.9	71.3	75.8	91.3	63
040_large_marker	—	—	85.3	97.2	97.1	**93**	97.6	100	92.2	**98**	100
051_large_clamp	—	—	52.2	75.3	67.4	26.4	72.6	33.3	**68.5**	**77.4**	69.6
052_extra_large_clamp	—	—	25.9	74.9	48.2	16.6	77.4	10.9	**25.3**	66.4	22
061_foam_brick	—	—	48.1	97.2	99.7	59	92	100	**92.9**	**98**	99.3

Table 4.2: Pose estimation accuracy per object class on the YCB-Video Dataset [139]. Best per class performance for ADD(-S) is in bold font. Ours achieves the best performance on a majority of object classes.

Performance for individual objects is shown in Table 4.2. We use the trained network of six objects provided by the authors of DOPE [123] and report the results. The ADD results are not available because the object coordinate frames used in the YCB object dataset [19], YCB video dataset for PoseCNN [139] and DOPE are different. As our method uses the frames from [139], and the transformation between [19] and [139] is not publicly available, we can not find the correspondence between model points required for ADD. We also applied ICP to DOPE pose estimates, but the performance was not improved. A possible reason is the sensitivity of ICP to the initial pose estimate. Figure 4.7 illustrates the input object segments and the overlay to the corresponding object model using the predicted 6D pose. With 256 points, although the input segment appears to be sparse, it is enough for capturing the surface geometry. We also denote the noise from imprecise segmentation and occlusion with arrows. From the overlay, we can see that the network can handle imprecise segmentation and occlusion. Some qualitative results are shown in Figure 4.8. Pose estimates from PC, DF, and our method are used for projecting object models onto 2D images.

[2]We apply ICP to DF results after iterative refinement.



(a) power drill (b) pitcher base

Figure 4.7: Illustration of input object segment transformed with accurate 6D pose estimates and overlay with the corresponding object model. Colored points represent the object segment (the color is for visualization only), and green points are the object model. The visible occlusion and segmentation noise are denoted with arrows.

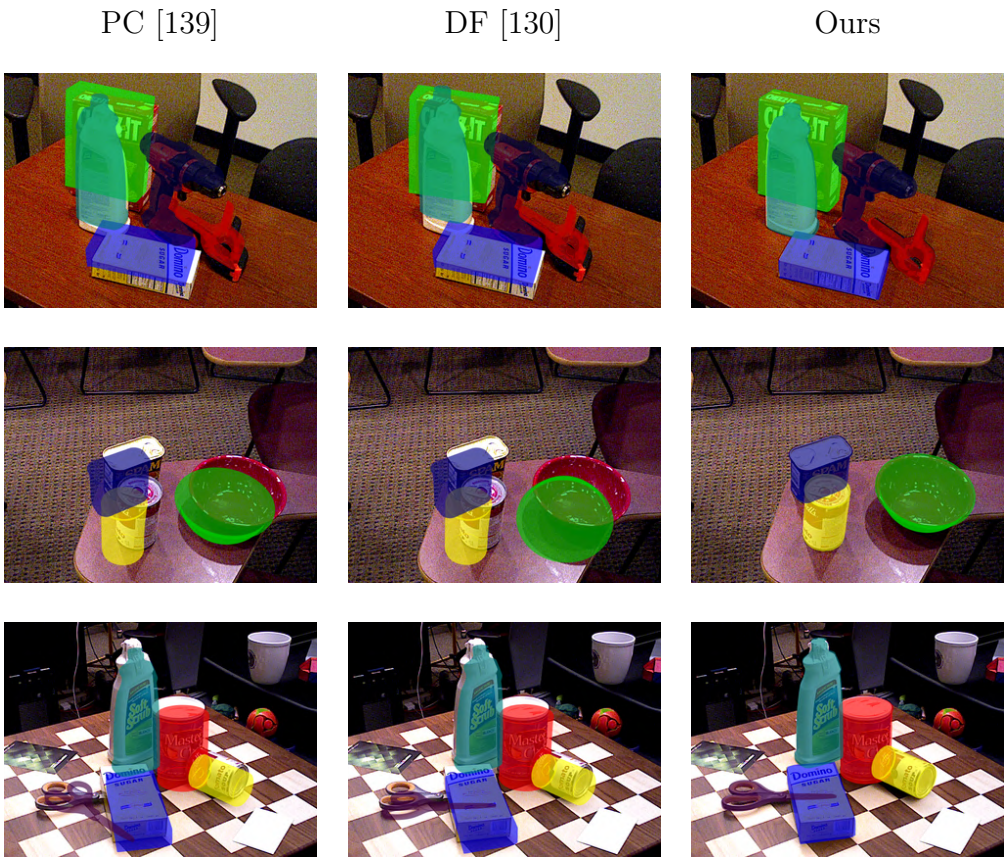


Figure 4.8: Qualitative results for 6D pose estimation. From left to right: PoseCNN (PC) [139], DenseFusion (DF) [130], and ours. The colored overlay indicates the predicted pose of the target object. Our method gives more accurate translation estimates, and also is able to give accurate rotation estimation for texture-less object (e.g. the red bowl).

	ape	bvise	cam	can	cat	driller	duck	e.box	glue	holep	iron	lamp	phone	avg
PC w/o ICP [139]	-	-	-	-	-	-	-	-	-	-	-	-	-	62.7
DF [130]	**92.3**	**93.2**	**94.4**	**93.1**	**96.5**	**87.0**	**92.3**	**99.8**	**100.0**	**92.1**	**97.0**	**95.3**	**92.8**	**94.3**
Ours w/o ICP	15.6	24.6	9.3	27.2	28.5	28.5	6.3	99.3	93.1	21.5	24.2	42.8	33	34.9
Ours	54.4	47.5	37.3	50.8	60.4	44.8	44.4	99.2	99.6	55.9	38	67.6	57.5	58.3
Ours w/o ICP (Color)	28.2	19.8	20.2	34.8	37.3	44.5	18.5	99.9	77.9	35.5	52.3	49.5	42.5	43.2

Table 4.3: Performance on the LineMOD dataset. E.box and glue are reported with ADD-S, others with ADD.

LineMOD dataset. Table 4.3 shows the evaluation result on LM. To follow the convention of other methods that report on this dataset, we report the ADD-S scores for egg box and glue, and report the ADD scores for the other objects. All methods use the same training data without additional synthetic data. We also compare with PC and DF. The result of PC is taken from [82]. Ours without ICP refinement performs poorly on this dataset. Even with pose refinement, the performance is still far from other methods.

This shows that this dataset is challenging for the depth-only method since both comparison methods use color information. One possible explanation is that since the training and testing data are taken from the same video sequence, learning the color feature from the train set is helpful for inference of the 6D object poses in the test set. To verify this hypothesis, we simply appended color information to the 3D coordinates and trained a network with both color and depth information. The results are in the last row of Table 4.3. It can be observed that using color does boost the performance, but the performance is still far from being competitive. It should be noted that this is not the most efficient way to use color information, and the color information is also very coarse (with only 256 color pixels). Another possible reason is that as reported in [68, 119], the intrinsic parameters are inaccurate, and the resulting pinhole camera model cannot provide an accurate depth estimation. This inaccurate depth information could also make the task more challenging.

One possible way to improve is to improve the amount of training data by adding synthetic data. The official LM dataset does not contain synthetic training data. On average, there are approximately 1200 frames for each object, and this split yields around 200 training frames and 1000 testing frames. For a deep learning system, this is a relatively small amount of training data. In Chapter 6, we will introduce a data synthesis pipeline for this purpose.

Occluded LineMOD dataset. We evaluate the same network used in the LM dataset also on the LMO dataset. Similar to the LM dataset, we report the ADD-S scores for egg box and glue, and report the ADD scores for the other objects. Table 4.4 shows the results. As expected, the performance of our method is also poor on this dataset.

	RGB	Depth	ape	can	cat	driller	duck	e.box	glue	holep	avg
PC w/o ICP	✓		9.6	45.2	0.93	41.4	19.6	22.0	38.5	22.1	24.9
PC	✓	✓	**76.2**	**87.4**	**52.2**	**90.3**	**77.7**	**72.2**	**76.7**	**91.4**	**78.0**
Ours w/o ICP		✓	1	2.6	0.1	1.8	0.8	72.8	46	2.2	15.9
Ours		✓	12.2	16.1	4.1	3.8	15.2	73.7	62.8	15.6	25.4

Table 4.4: Percentage of correctly estimated poses on Occluded LineMOD. E.box and glue are reported with ADD-S, others with ADD.

4.5.3 Occlusion

For a given target object in a frame, the occlusion factor O of the object is defined as [38]

$$O = 1 - \frac{\lambda}{\nu}, \tag{4.12}$$

where λ is the number of pixels in the 2D ground truth segmentation, and ν is the number of pixels in the projection of the 3D object model onto the image plane using the camera intrinsic parameters and the ground truth 6D pose, when we assume the object would be fully visible. The occlusion factor of the YCB-Video dataset ranges from 0.8% to 87%. Figure 4.9 shows the histogram of occlusion factor. Figure 4.10 shows examples of different occlusion factors, the target objects are denoted with green masks.

We divide this range into 8 bins with a bin width of 10% and report the prediction accuracy (ADD-S) with a threshold of 1 cm. Figure 4.11 illustrates the results. It can be observed that our method (Ours) has competitive performance when the occlusion is lower than 40%, then both ours and PC start to suffer as the amount of occlusion increases. One possible reason is that DF outputs per-pixel prediction with confidence scores, while ours and PC provide only one pose prediction. This per-pixel prediction may have helped to provide better performance when the amount of occlusion is higher than 40%.

4.5.4 Ablation Study

Network architecture. To investigate whether translation and rotation should be regressed with the same or separate networks, we compare the performance of different architectures. We alter the network architecture by incrementally sharing the layers between translation and rotation networks. Table 4.5 shows the results in terms of ADD, ADD-S, and accuracy for translation and rotation under certain thresholds. None denotes the proposed architecture which regresses translation and rotation with two separate networks. The numbers in the first column denote the number of shared layers between translation and rotation BaseNet. We compare performance without ICP refinement. When sharing layers, the performance is worse than using two separate networks.

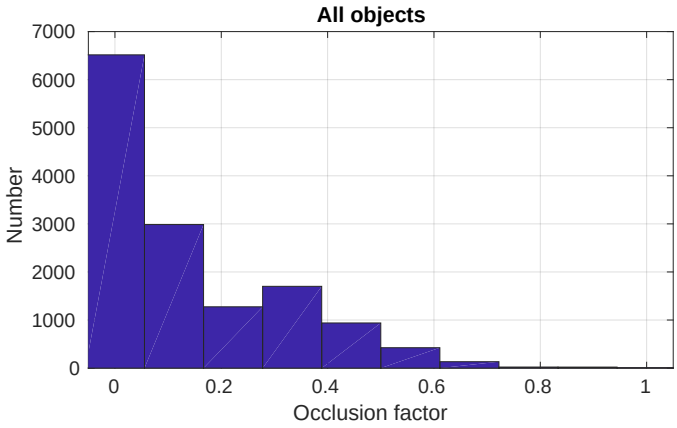


Figure 4.9: Histogram of occlusion factor of 2,949 key frames in YCBV test dataset.

(a) no occlusion, with occlusion factor 0.02

(b) moderate occlusion, with occlusion factor 0.45

(c) high occlusion, with occlusion factor 0.87

Figure 4.10: Illustration of different degrees of occlusion.

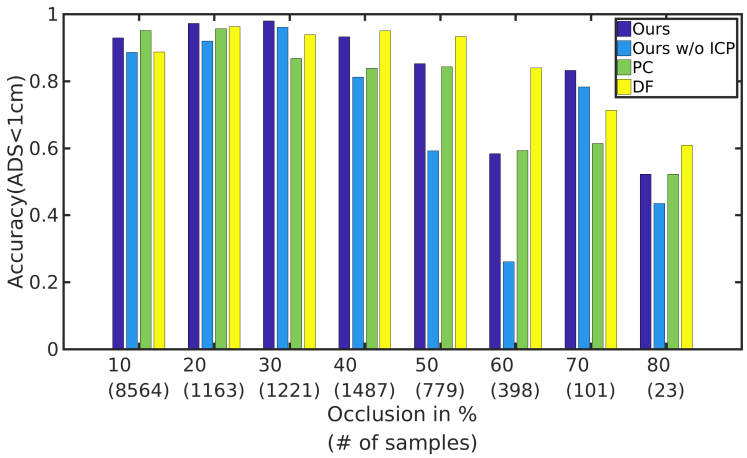


Figure 4.11: Effect of occlusion compared to PC [139] and DF [130]. The horizontal axis denotes the upper limit (occlusion in %) of each bin. Width for each bin is 10%. Numbers in parentheses denote the number of samples in corresponding bin. Ours is competitive with state-of-the-art methods when occlusion is lower than 40%.

shared layers	ADD	ADD-S	rot_err<10°	tran_err<1 cm
none	**76.0**	**91.3**	**41.3**	**73.0**
1	75.2	91.1	39.5	72.0
2	75.5	91.1	38.6	69.8
3	75.1	91.1	41.2	69.6
4	75.5	91.2	39.4	70.3
5	75.2	91.1	39.1	69.7
all	63.0	87.8	23.3	69.3

Table 4.5: Accuracy with different network structures. Best performance is in bold font. Using two separate networks performs best.

Rotation representation	<10°	<15°	<20°
Axis-angle	**41.3**	**52.7**	**59.6**
Quaternion	37.1	48.5	56.7

Table 4.6: Different rotation representations. Geodesic distance is used as the loss function for both cases.

Loss function	<10°	<15°	<20°
Geodesic	**41.3**	**52.7**	**59.6**
L2	40.8	52.5	58.4

Table 4.7: Different loss functions for rotation regression. Axis-angle representation is used for both cases.

We also tested an architecture that shares all the layers while having the same number of parameters as the proposed structure with a doubled layer width. The performance is similar to the architecture with the single width, and this verifies that the performance deterioration is not caused by insufficient network capacity. This result verifies that using separated networks for translation and rotation is a more suitable design choice.

Rotation representation and loss function. We investigate the impact of different rotation representations and loss functions. For comparing quaternion to axis-angle, we adapted our rotation network to have 4-dimensional output instead of 3. The output is normalized and then converted to the axis-angle representation. For comparing L2 loss with Geodesic distance, we keep the rotation representation in axis-angle format and apply different loss functions. Table 4.6 shows the accuracy of rotation prediction with different thresholds. With the same loss function, using axis-angle yields a better result than the quaternion. This indicates that the axis-angle is a better choice for rotation learning. Table 4.7 shows that, with the same rotation representation, geodesic loss outperforms $L2$ loss. Since geodesic distance also has a better mathematical justification, this makes it a better choice. It is

worth noting that the axis-angle representation has discontinuities at 180°, which may hurt the system performance. Hence, addressing this issue can potentially improve the system performance. However, this is out of the scope for this thesis and more insights regarding this can be found in [152].

4.5.5 Active Points

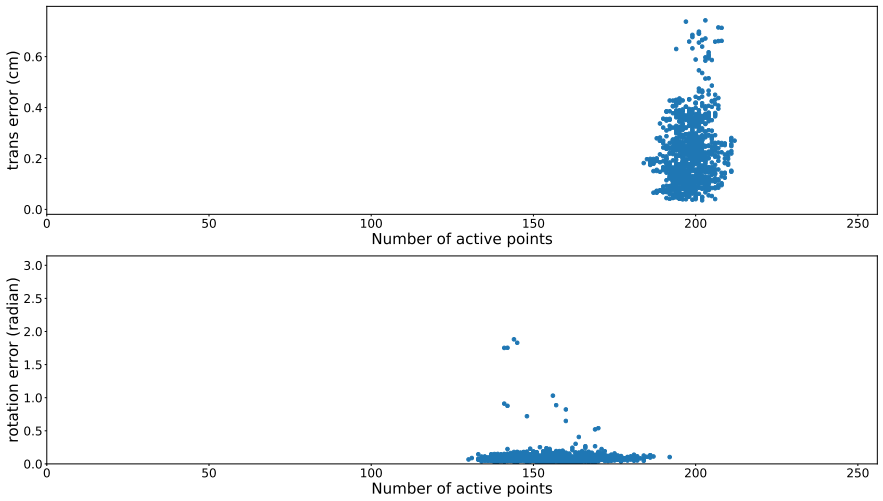
With the max pooling operation, only the points which have the max value for one or many feature dimensions are considered for final pose inference. This means only a portion of the input segment is contributing to the final pose estimation. We name those points the "active points".

We investigate how many active points the translation and rotation BaseNet have. On average, out of 256 points, the translation BaseNet uses around 200 points for translation regression. The rotation BaseNet uses around 150 points for rotation regression. We show the relationship between the number of active points and translation, rotation error, respectively (Figure 4.12). It can be observed that the number of active points is in the same range for those objects. For non-rotational symmetric objects such as power drill and banana, the rotation error is low on average. On the contrary, with the rotational symmetric master chef can, the rotation error varies a lot. This is because our method does not deal with rotational symmetry.

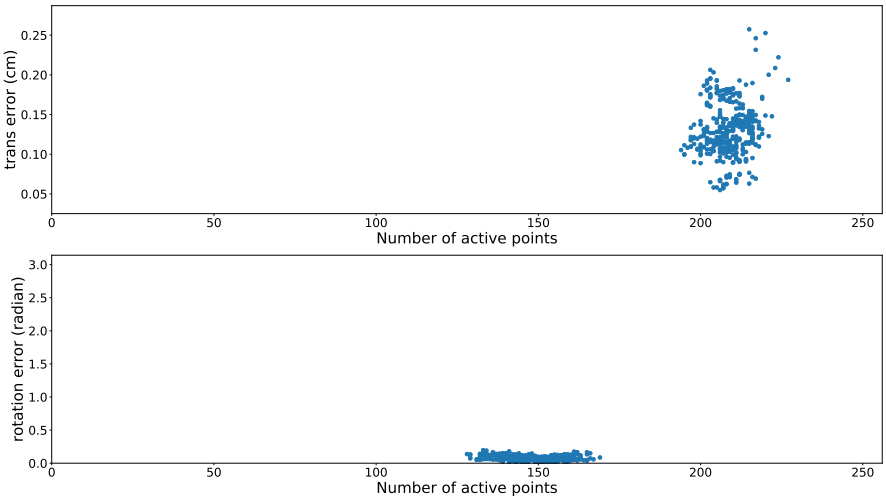
Figure 4.13 shows some examples for active points. All point cloud segments have 256 points. For each point cloud segment, the active points are denoted with red and the others with black. The object model is shown in cyan wire-frame in the object coordinate, and the point cloud segment is transformed into object coordinate with ground-truth poses. For additional information, the estimation rotation and translation errors are provided in the captions. The segments presented here have segmentation noise as well as occlusions. In general, the active points are relatively uniformly distributed in the segment, and there are no particular regions on the objects that are more active than others. It is interesting to see that the networks also consider the background points for pose inference. This implies that the networks also rely on the "errors" made by the segmentation methods. This is not desired because this means if the segmentation methods used for the training and testing phase are different, the networks may perform poorly during the testing phase.

4.5.6 Time Performance

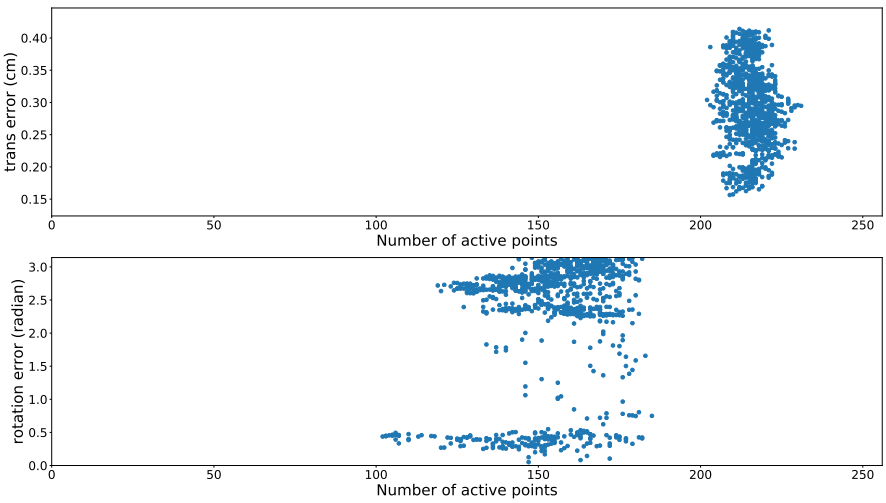
We measure the time performance on an Nvidia Titan X GPU. The system is implemented with Tensorflow. Training the proposed model takes approximately 22 hours. Pose estimation by a forward pass through our network takes 0.11 seconds for a single object. The 10 iterations of ICP refinement require an additional 0.3 seconds.



(a) power drill



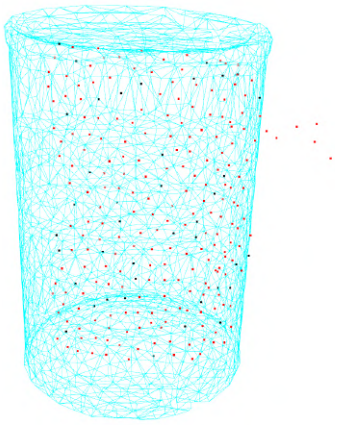
(b) banana



(c) master chef can

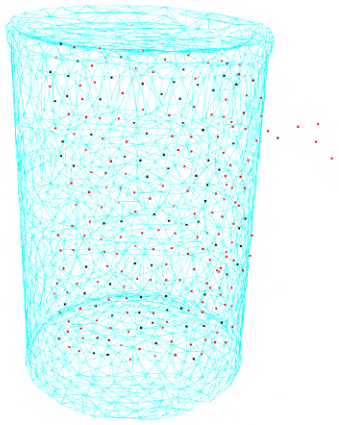
Figure 4.12: Illustration of number of active points and pose estimation errors.

Translation

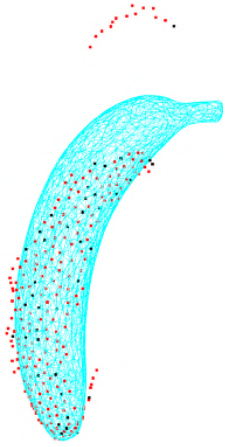


(a) 216 active points, error 0.4 cm

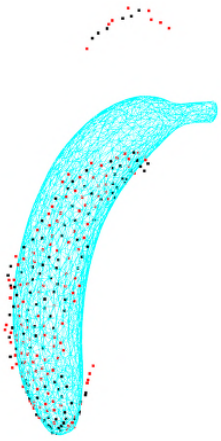
Rotation



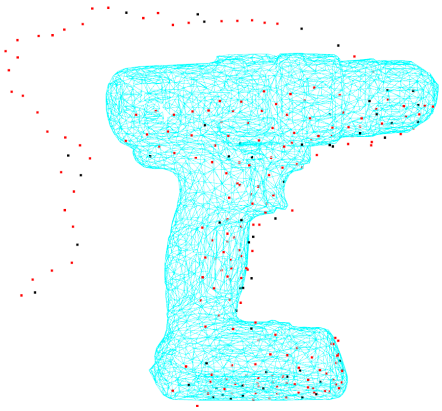
(b) 175 active points, error 2.27 radian



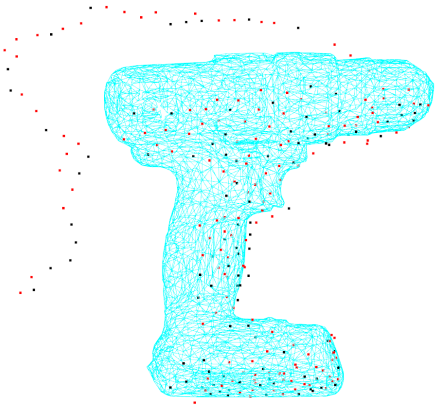
(c) 217 active points, error 0.14 cm



(d) 137 active points, error 0.07 radian



(e) 201 active points, error 0.4 cm



(f) 152 active points, error 0.1 radian

Figure 4.13: Examples of active points. Active points are denoted with red, and other points in the object segment with black. The object model is drawn with the cyan wire frame.

4.6 Summary

In this chapter, we have proposed a system for fast and accurate 6D pose estimation of known objects. We formulate the problem as a supervised learning problem, use two separate networks for rotation and translation regression, and use point clouds as input for the regression. We use axis-angle as rotation representation and geodesic distance as the loss function for rotation regression. Ablation studies show that these design choices outperform the commonly used quaternion representation and L2 loss. Experimental results show that the proposed system outperforms two state-of-the-art methods on a public benchmark at the time of this work. This work verifies that features extracted from point clouds with deep networks can be used for accurately regressing the object pose.

From the perspective of system building blocks, one major limitation of this work is that the PointNet-like structure can not capture information about the local neighbourhood. It is interesting to investigate whether structures such as PointNet++ (Section 2.3.2) and EdgeConv (Section 2.3.3) can give better performance. Furthermore, although it performed well on the YCBV dataset, the evaluation results on LM and LMO are really poor. As mentioned in Section 2.4.4, the accuracy of depth information in LM is poor, which might be one of the causes for this poor performance of our depth-based method. In the next chapter, we attempt to improve those aspects by investigating a hybrid structure, which is a mixture of fully supervised and self-supervised learning for this task.

Chapter 5

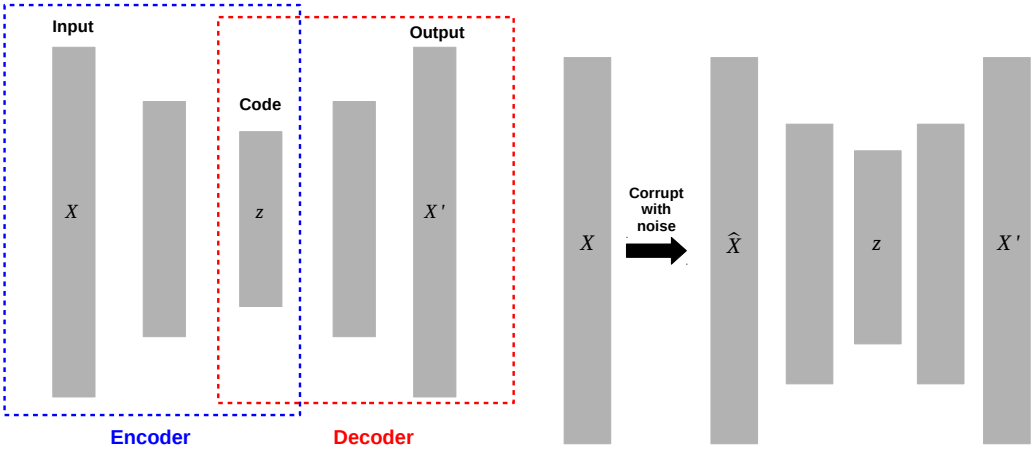
CloudAAE: Learning 6D Object Pose Regression with an Augmented Autoencoder

In the previous chapter, we presented a simple and effective framework that regresses 6D object poses from depth information represented by point clouds in a supervised learning manner. In this chapter, we investigate a hybrid framework that combines supervised learning and self-supervised learning for the same task. For the supervised learning aspect, we follow a similar scheme as the previous chapter and regress to 3D rotation and 3D translation using deep features from point clouds. For the self-supervised learning aspect, we use an autoencoder and conduct a reconstruction task. Specifically, we use an augmented autoencoder for learning a latent code that encodes 6D object pose information for pose regression. The latent code of the autoencoder contains the pose information, and it is used as the input to two pose regressing networks. We present a new framework for regressing the 6D object pose from point cloud segments. A point cloud-based augmented autoencoder is used to learn a latent code that encodes object pose information. This code is used for regressing the 6D object pose.

We introduce the motivation of using an autoencoder structure in Section 5.1. The system architecture is present in Section 5.2. Loss functions used for training and testing the proposed system are described in Section 5.3. Experiment results are shown in Section 5.4 and Section 5.5 provides the concluding remarks and discussion for this chapter.

5.1 Why Autoencoder?

Recalling from previous chapters, we are dealing with the task to estimate a 6D object pose from an input point cloud segment with class information. The input point cloud segment has a dimension $n \times (3 + k)$, in which n is the total number of points and k is the total number of classes. For example, assuming we have 256 points and 21 classes (as for the YCBV data set), then the total input data



(a) Autoencoder (b) Denoising Autoencoder

Figure 5.1: Illustration of (denoising) autoencoder.

dimension is 6144. This is approximately 1000 times bigger than the output dimension, which is 6 (3D rotation and 3D translation). Hence, the 6D object pose information is very sparsely encoded in the input point cloud data. This makes it difficult for the network to extract the essential information about the object pose.

One classic approach to deal with this problem is dimension reduction [72]. In the above example, the input data dimension is 6144, and it is referred to as the superficial dimensionality. The task of dimension reduction is to extract a feature with a smaller intrinsic dimensionality from the input with minimal loss of information [72]. In machine learning, this can be achieved by using neural networks with "encoder & decoder" structure or generally referred to as autoencoder (AE). The key trait of an autoencoder is that it contains a "bottleneck" layer, which is of smaller dimension than either the input or the output [72]. This bottleneck layer is often referred to as latent code or latent representation. By training the network to perform identity mapping and approximating the input at the output layer, the latent code learns to encode the essential information about the input. Figure 5.1(a) shows an illustration of an autoencoder presented with 3 fully connected hidden layers. Input and output are denoted with X and X', respectively. The hidden layer in the middle is the latent code z.

However, the risk of learning an identity mapping is that there is an obvious solution by "simply copying the input" [128], especially when the dimension of the "bottleneck" is large. In this case, the network does not learn anything useful but an identity function. One way to tackle this is to make the reconstruction more challenging by corrupting the input and let the decoder output the clean input, or "denoises" the corrupted input. This kind of autoencoder is referred as the denoising autoencoder (DAE). In this case, since the latent code facilitates the reconstruction of the clean input, it is invariant to the noise used for corrupting the input [119]. Figure 5.1(b) is an illustration of denoising autoencoder, where the input to the network is $\hat{X}$, and it is obtained by adding noise to X. As mentioned in Section 3.1.2, Sundermeyer et al. proposed the augmented autoencoder, which

is a generalization of a denoising autoencoder [119]. The idea of an augmented autoencoder is that by applying augmentations to the input, the encoding becomes invariant to those augmentations. Meanwhile, by setting the reconstruction goal accordingly, it is possible to control what information the latent code encodes. In general, by using an AAE, it is possible to control which properties the latent code encodes and which properties are ignored [119].

In this chapter, we propose to adapt the augmented autoencoder proposed by Sundermeyer et al. [119] for a point cloud-based system. We focus on learning a latent code that encodes the 6D pose information. The task of the AAE is to reconstruct a point cloud segment in the desired 6D pose. Moreover, this segment is noise and occlusion free. In this way, the latent code contains the necessary information for regressing the object pose. Using the latent code as the input, we use two separate networks for regressing 3D rotation and 3D translation, as suggested by the previous chapter. There are three main differences between our approach and [119]. First, they use a 2D color image as input, while ours uses point clouds. Second, our approach does not require creating off-line codebooks. Third, we use the latent code from the AAE for both 3D translation and 3D rotation estimation. Instead of using the latent code from AAE for nearest neighbor search with a codebook, we use it for direct 6D pose regression with two separate networks. Regarding the "augmented autoencoder", we focus on introducing the "autoencoder" aspect by describing the system structure and its performance in this chapter. The "augmented" aspect will be investigated in the next chapter when introducing our data synthesis pipeline.

5.2 System Architecture

We introduce our data synthesis pipeline and an augmented autoencoder based 6D pose estimation system, referred to as CloudAAE. CloudAAE is a multi-class system, which means we use the same system to predict poses for objects from different classes. Given a known object represented by a set of points in the camera coordinate C, a 6D pose estimation system aims to find the translation $\mathbf{t}$ and rotation R that describes the transformation from the object coordinate system O to C.

Figure 5.2 shows an overview of CloudAAE. Similar to the input of CloudPose (Chapter 4), assuming we have a depth image and its corresponding semantic segmentation, it is possible to obtain the class and point cloud segment of a target object. The target objects are known objects, and we have a 3D object model of them. We denote the point cloud segment as P^C. Before inputting to the AAE, we normalize P^C by removing its coordinate mean $\mu_{\mathbf{p}}$ and denote the normalized segment as P^N. Moreover, P^C often contains some sensor noise or only represents a partially visible object due to external occlusion. The AAE is expected to output a noise and occlusion free object segment ΔP^C_{recon} at the same 6D pose as P^N. P^C_{recon} is obtained by adding back $\mu_{\mathbf{p}}$ to ΔP^C_{recon}. On the other hand, constrained by the reconstruction task, the latent vector encodes the 6D object pose information.

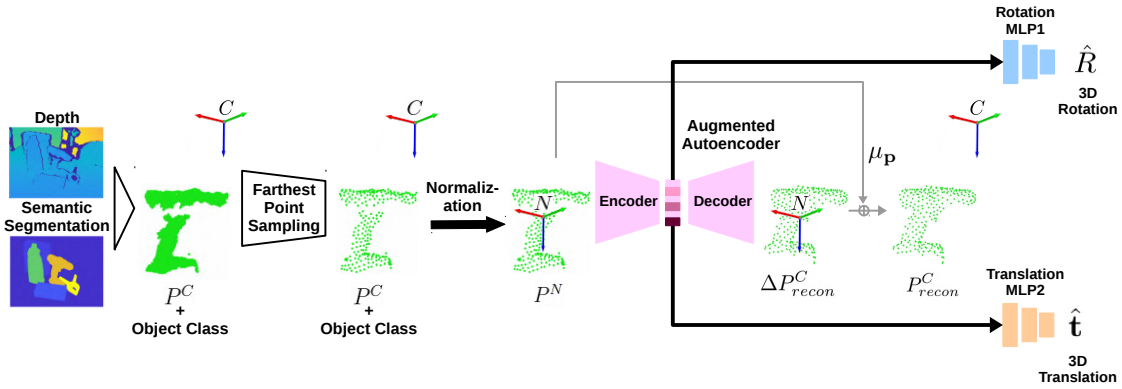


Figure 5.2: The class information and point cloud segment P^C of a target object is obtained from a pair of depth and semantic segmentation images. After down-sampling P^C with FPS, it is normalized by removing its coordinate mean $\mu_{\mathbf{p}}$. The resulting P^N and the corresponding class information are used as the network input. The expected ΔP^C_{recon} is a noise and occlusion free segment. By adding $\mu_{\mathbf{p}}$ to ΔP^C_{recon}, we obtain P^C_{recon} at the desired 6D pose. Meanwhile, the latent code is used with two separate 3-layer multi-layer perceptrons (MLP1 and MLP2) for regressing 3D rotation $\hat{R}$ and 3D translation $\hat{\mathbf{r}}$.

The latent code is used as the input to two separate pose regressing networks for regressing 3D rotation $\hat{R}$ and 3D translation $\hat{\mathbf{t}}$. Overall, CloudAAE contains an augmented autoencoder and the 6D pose regressors.

Augmented autoencoder (AAE) We adapted the idea of AAE proposed in [119] for point clouds and use an adapted version of the Dynamic Graph (DG) [132] as the encoder. We also evaluated different choices for the encoder and the ablation study results are presented in Section 5.4.3. Figure 5.3 illustrates the architecture of our point cloud-based AAE. The AAE consists of an encoder and a decoder. The 3D coordinate of each point in P^N_{occ} is concatenated with one-hot class information. This is used as the input to the encoder. The encoder computes a latent vector, and the desired output from the decoder is a noise and occlusion free segment in the 6D pose defined by R and $\mathbf{t}$. The input is of dimension $n \times (3 + k)$, in which 3 represents 3D coordinates, and k is the total number of classes.

Our encoder is an adapted version of Dynamic Graph, which is a deep network processing unordered point sets and is introduced in Section 2.3.3. Recall that, in contrast to the PointNet structure [103] we used for CloudPose (Chapter 4), where all features are extracted based only on a single point, DG explicitly considers the local neighbors of individual points. For each point $\mathbf{q}_i$ in a point set, a k-nearest neighbor graph is calculated. In all our experiments, we empirically use $k = 10$. For the edge function, we use an MLP layer with shared weights for each edge feature. To consider all edge features information for the reconstruction task, we use average pooling as the aggregation operation. The ablation study on choosing the aggregation operation is presented in Section 5.4.3. This EdgeConv is

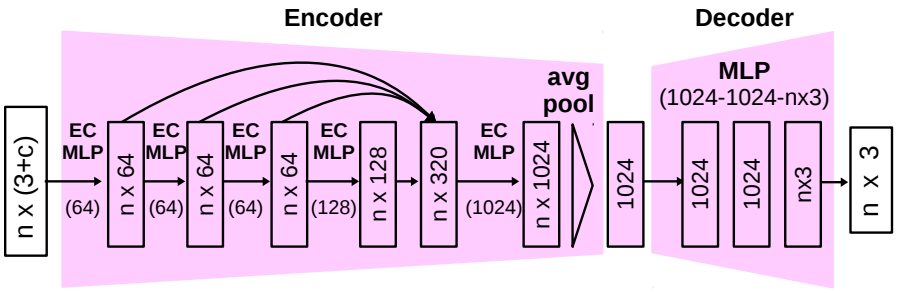


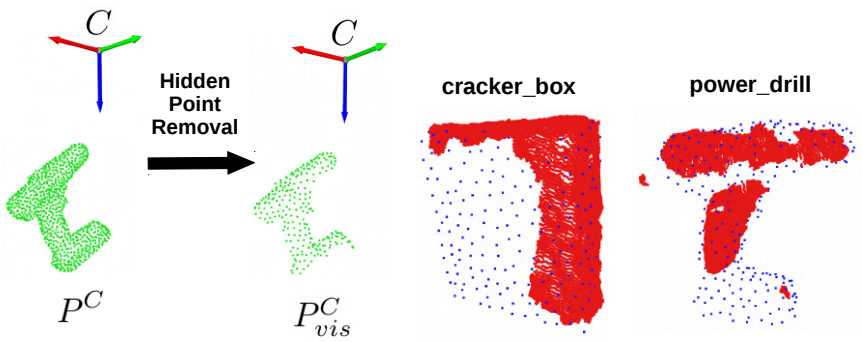
Figure 5.3: The point cloud based Augmented Autoencoder. The numbers of neurons in EC-MLP and MLP layers are indicated in parentheses. EC is short for EdgeConv. Dimensions of intermediate features are indicated without parentheses. Skip connections denote the concatenation of edge features. A 1024-dimensional feature vector for each point with its local neighbors is learned with shared weights for the encoder. An average pooling layer aggregates the individual features into a 1024-dimensional latent code. Finally, three fully-connected layers output noise and occlusion-free point cloud segment.

repeated, calculating the nearest neighbor graph for the feature vectors of the first shared MLP layer and the subsequent layers. Finally, the edge features from each EdgeConv are concatenated and processed with an MLP layer. This concatenation is illustrated with skip connections in Figure 5.3. A 1024-dimensional feature vector is learned for each point. These features are average pooled to obtain a global representation of the input point cloud segment. We use four EdgeConv layers on edge features and one EdgeConv layer on the concatenated edge feature. The encoder outputs a latent code with a dimension of 1024. The decoder contains 3 fully connected layers with dimensions of 1024, 1024, and $n \times 3$.

6D pose regressors We additionally add two networks for regressing the 6D poses. Since the decoder is able to reconstruct the segment in the same 6D pose from the latent code, the latent code contains the object pose information. Hence, the latent code is used as the input to two networks for regressing 3D rotation and 3D translation, respectively. Each network contains three MLP layers with dimensions 512, 256, and 3, respectively.

5.3 Loss Function and Testing Phase

We normalize P_{occ}^{C} by removing its mean $\mu_{\mathbf{p}}$ before inputting to the AAE (Figure 5.2). The output of AAE is the residual 3D coordinates of the reconstructed point cloud segment. The output of translation prediction is the residual of translation. The full 3D coordinates and 3D translation are obtained by adding back $\mu_{\mathbf{p}}$.



(a) (b)

Figure 5.4: Illustration of the learning target and the output of AAE. (a) The learning target P^C_{vis}. (b) Examples of AAE output for objects in the YCB video dataset during testing. The input to the AAE is denoted in red, and the reconstructed noise free and occlusion free point cloud segment are denoted in blue.

Augmented autoencoder (AAE) Assuming the AAE outputs residual 3D coordinates ΔP^C_{recon}, the full 3D coordinate is $P^C_{recon} = \Delta P^C_{recon} + \mu_{\mathbf{p}}$. The learning target P^C_{vis} is obtained by applying hidden point removal (HPR) to P^C without introducing any occluders, as illustrated in Figure 5.4(a). We will describe HPR in detail in Section 6.2.3. The Chamfer distance [28] is used to measure the difference between P^C_{recon} and P^C_{vis}. We also tried the Earth Mover's distance [28], which is more computationally expensive and gives a similar performance. The difference between P^C_{recon} and P^C_{vis} is:

$$\begin{aligned} l_{CD}(P^C_{recon}, P^C_{vis}) = \frac{1}{m} \sum_{\mathbf{x} \in P^C_{recon}} \min_{\mathbf{y} \in P^C_{vis}} \|\mathbf{x} - \mathbf{y}\|_2 \\ + \frac{1}{n} \sum_{\mathbf{y} \in P^C_{vis}} \min_{\mathbf{x} \in P^C_{recon}} \|\mathbf{y} - \mathbf{x}\|_2. \end{aligned} \tag{5.1}$$

The number of points in P^C_{vis} and P^C_{recon} are denoted by m and n, respectively. The desired P^C_{recon} is a denoised, unoccluded object segment. Figure 5.4(b) shows examples of P^C_{recon} output by a trained AAE.

6D pose regressors For rotation regression, the axis-angle representation is used as the regression target. An axis-angle is a vector $\mathbf{r} \in \mathbb{R}^3$ that represents a rotation of $\theta = \|\mathbf{r}\|_2$ radians around the unit vector $\frac{\mathbf{r}}{\|\mathbf{r}\|_2}$. Geodesic distance is used as the loss function for measuring the distance between prediction $\hat{\mathbf{r}}$ and ground truth $\mathbf{r}$ [38]. With $\hat{R}$ and R being the corresponding rotation matrices for $\hat{\mathbf{r}}$ and $\mathbf{r}$ respectively, the *rotation loss function* $l_r(\hat{\mathbf{r}}, \mathbf{r})$ is defined as

$$l_r(\hat{\mathbf{r}}, \mathbf{r}) = \arccos\left(\frac{\text{trace}(\hat{R}R^T) - 1}{2}\right). \tag{5.2}$$

The regression target for translation is the residual of translation. Given $\widehat{\Delta \mathbf{t}}$ as the translation residual, full translation prediction is $\hat{\mathbf{t}} = \widehat{\Delta \mathbf{t}} + \mu_{\mathbf{p}}$. With $\mathbf{t}$ being the ground truth translation, the *translation loss function* $l_t(\hat{\mathbf{t}}, \mathbf{t})$ is measured with $L2$-norm $l_t(\hat{\mathbf{t}}, \mathbf{t}) = \left\| \mathbf{t} - \hat{\mathbf{t}} \right\|_2$.

Total loss function The total loss l_{total} used for training the AAE and pose regressors is the combination of above losses:

$$l_{total} = \alpha l_{CD} + \beta l_t + l_r, \tag{5.3}$$

where α and β are scaling factors.

Testing phase In the testing phase, the trained CloudAAE is used for estimating 6D object poses. If the input is an RGB-D image, semantic segmentation is used to obtain the object segment and the corresponding class information. After segmenting the depth image, the point cloud is obtained using the depth value and the intrinsic camera parameters. The point cloud segment is normalized and concatenated with one-hot class information. This is used as the CloudAAE input. As semantic segmentation is a well-studied topic [4], we assume the object segment and class information is provided by an off-the-shelf method. After obtaining the pose estimates, we further refine the poses with ICP.

5.4 Experiments

We evaluate the proposed system on the LineMOD [52], Occluded LineMOD [14] and YCB-Video [139] datasets. Besides the prediction accuracy (Section 5.4.2), we also investigate the impact of using different network structures as the encoder and different dimensions of the latent code (Section 5.4.3). We further investigate whether the reconstruction error of AAE can be used as an uncertainty measure of the network pose prediction (Section 5.4.4). The time performance is presented in Section 5.4.5.

5.4.1 Experiment Setup

The experiment setup is mostly the same as we have in Chapter 4. The Adam optimizer is used during training with a learning rate of 0.0008. The training batch size is 128. The number of points of the input point cloud segment is $n = 256$. For the total loss, we use $\alpha = 1000$ and $\beta = 10$, which is given by the ratio between the expected errors of the reconstruction, translation, and rotation at the end of the training [36]. Batch normalization is applied to all layers, and no dropout is used. For refinement, we use the Point-to-Point ICP registration provided by Open3D [151] and refine for 10 iterations. Similarly, the initial search radius is 0.01 m and is reduced by 10% after each iteration. DenseFusion, CloudPose, and pur CloudAAE use object segmentation provided by PoseCNN during testing. The training details regarding the dataset split are the same as described in Section 4.5.1.

	RGB	D		RGBD		
	PoseCNN[1] [139]	CloudPose [36]	CloudAAE [35]	PoseCNN [139]	DenseFusion [131]	PVN3D [49]
LM	62.7	58.3	86.7	-	94.3	-
LMO	-	25.4	54.9	78.0	-	-
YCBV	-	94.7	94.0	-	93.2	96.1

Table 5.1: Performance on LM, LMO and the YCBV dataset. For LM and LMO, we report the average AUC in the case that E.box and glue are reported with ADD-S, others with ADD. We report the AUC of ADD-S metric on YCB-Video dataset. The result for the RGB case of PoseCNN is without ICP refinement, all the other results are with ICP refinement.

5.4.2 Prediction Accuracy

The comparison results on all three datasets are summarized in Table 5.1. We will discuss those results in this section.

YCBV We compare the performance with the state-of-the-art methods DenseFusion [130] and PVN3D [49], which is the most recent state-of-the-art method. Moreover, we compare with CloudPose, which is the framework we proposed in Chapter 4. We follow the convention in [49] to show only the ADD-S scores for comparison. Since the YCBV dataset contains mostly rotational symmetric objects, this measure is more representative compared to ADD. In general, using only depth for pose inference, Ours still shows comparable performance to the RGB-D based performance. On the other hand, Ours has a similar performance as our previous system CloudPose. The slight performance drop of Ours can be caused by sharing a certain number of layers for 3D rotation and 3D translation regression. Although we dedicated two separate networks for rotation and translation regression, since they both use the latent code as the input, they do share the network layers in the encoder. As shown in the ablation study in the previous chapter (Table 4.5), sharing some layers does hurt the performance.

LM We compare the performance with PoseCNN [139] and DenseFusion [130]. PoseCNN without ICP uses only color information for pose inference. DenseFusion uses both color and depth information for pose inference. We report the ADD-S scores for the egg box and glue, and ADD scores for the other objects. Using only depth for pose inference, Ours achieves an average estimation accuracy of 86.7%. Although this is still around 8% performance difference comparing to DF (94%), it is a huge improvement comparing to CloudPose (58.3%). This shows the effectiveness of using the latent code as the input to both rotation and translation

[1]Result is without ICP refinement.

regressors. The latent code contains much condensed information about the input segment, and it is instructed to encode 6D object pose information using the AAE.

LMO We compare the performance with PoseCNN [139]. Like LM, we report the ADD-S scores for the egg box and glue, and ADD scores for the other objects. The results on this dataset show a similar story as the LM dataset. For this dataset, Ours has a larger performance gap with RGB-D based PC (54.9% comparing to 78.0%). One potential reason for the large gap is that the occlusion is more challenging to handle with only depth information. On the other hand, the performance of Ours achieves more than two folds of improvement compared to CloudPose (54.9% comparing to 25.4%). This again verifies the efficiency of using an augmented autoencoder based system.

5.4.3 Ablation Study

In this section, we first investigate the impact of using different network structures as the encoder. We then show how the size of the latent code impacts the system performance. We also use a different aggregation operation on the edge features to see how the performance is impacted.

Encoder Recalling from Section 2.3, there are three popular deep networks for learning on point clouds. They are PointNet (PN), PointNet++ (PN++) and Dynamic Graph (DG). The main difference among those structures is that the PN does not explicitly consider the neighborhood information of input points, while the other two consider it. In our previous experiments, we used DG as the encoder. This experiment replaces DG with PN and PN++, respectively, and trains the networks until convergence. The performance results are shown in Table 5.2. Overall, using DG shows the best performance. However, although PN does not consider neighbourhood information, the performance is only around 3% worse. Given its much lighter structure and shorter training time, PN can be viewed as a competitive candidate for use as the encoder. On the other hand, it is surprising that the performance is significantly worse when using PN++ as the encoder. One significant difference between PN++ and DG when considering the neighbour information is that PN++ simply locates neighbourhood points and uses a PN structure for feature extraction, while DG explicitly constructs edges between the center point and its neighbour points for feature extraction. Moreover, we use the default parameters for the range of the nearest neighbour for PointNet++, which might be suboptimal for our task. More investigation in setting proper parameters may improve the performance.

Latent code size We also investigate how the performance is impacted by the latent code size. We train networks with different sizes for the latent code. All the networks are trained on the same synthetic data and trained until convergence. The results without ICP are shown in Table 5.3. The generalization ability of

encoder	PN	PN++	DG
avg	70.7	49.6	**73.6**

Table 5.2: Results (w/o ICP) with different encoders on LM.

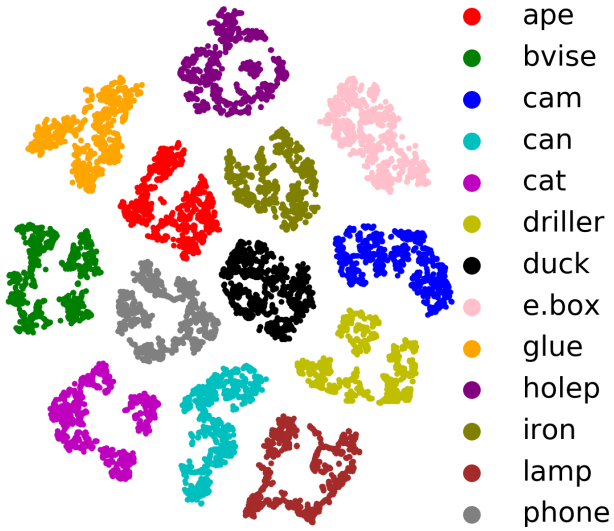


Figure 5.5: T-SNE visualization of latent code for the LineMOD dataset.

Dimension	512	1024	2048	4096
avg	69.5	**73.6**	73.4	72.4

Table 5.3: Results (w/o ICP) with different latent code sizes on LM.

a network is the best when the latent code is of size 1024 (73.6%), while the performance with size 2048 is also good (73.4%). With a smaller size of 512, it seems the latent code is too small to capture all the required information for pose estimation. The performance is also decreasing with size 4096 (72.4%.) We pick 1024 for our system. We show a t-SNE [126] visualization of the latent code for the LineMOD dataset in Figure 5.5. The t-SNE is short for t-distributed stochastic neighbor embedding and it is a method for visualizing data in high-dimensional space by giving each data point a location in a low-dimensional space of two or three dimensions. The similarity and dissimilarity of data point in the high-dimensional space are preserved in the low-dimensional space. As it can be seen from the figure (5.5), the latent codes are well clustered for each class.

Aggregation operation The operation we used for aggregating edge features is average pooling. The intuition is to summarize all the information instead of picking only a subset for reconstruction. To verify this hypothesis, we replace all the average pooling with max-pooling and train a network. We show the performance

aggregation function	max pooling	average pooling
avg	64.6	**73.6**

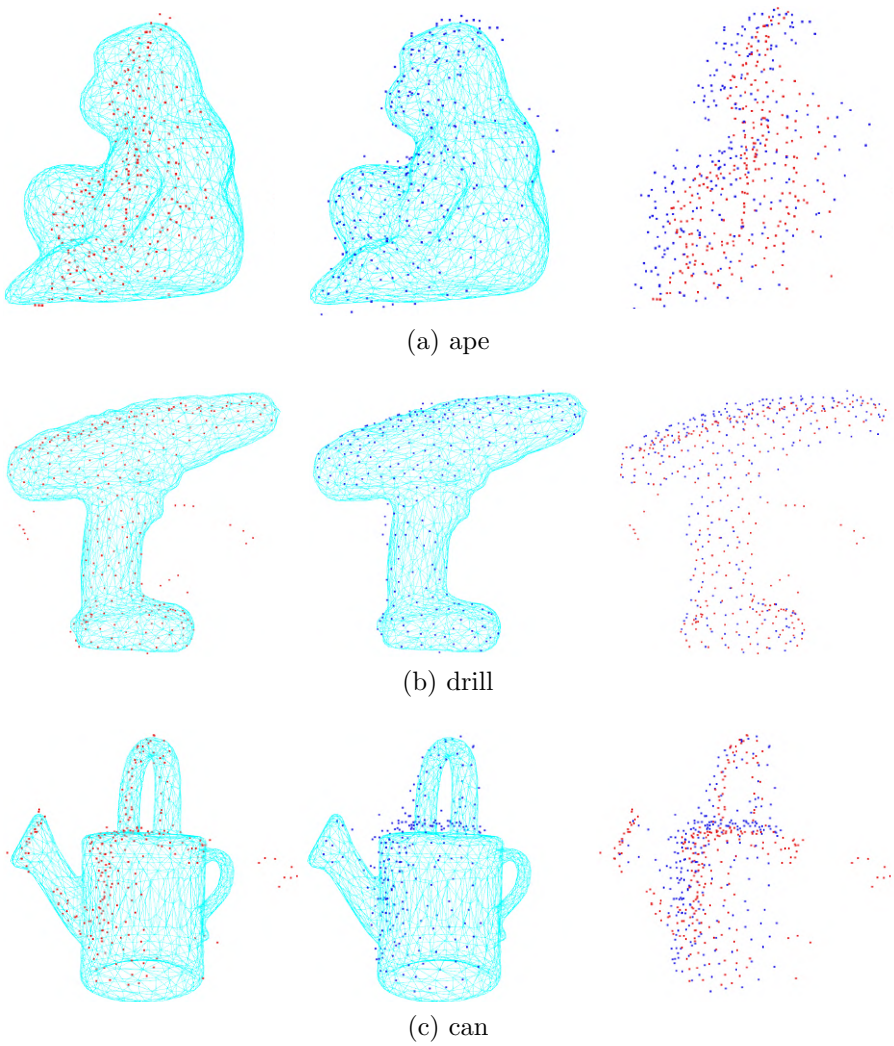
Table 5.4: Results (w/o ICP) with different aggregation functions on LM.

in Table 5.4. With max pooling, the performance is dropped by almost 10% (from 73.6% to 64.6%). This verifies our hypothesis that it is not optimal to use only a subset of edge features for reconstruction and pose estimation tasks.

5.4.4 Reconstruction and Confidence Measure

We want to investigate whether the reconstruction output can be used as an uncertainty measure for the 6D pose estimation. Since we have the input segment and the "clean" reconstruction of the input segment, we can measure the difference between those two segments and check its correlation with 6D pose errors. Figure 5.6 shows some examples of the input segment (in red) and the corresponding reconstructed segment (in blue). All segments are transformed with the ground truth 6D pose to be superimposed on the object model, which is shown in the cyan wire frame. It can be seen that with the ape, the input depth is deviated from the object surface due to inaccurate camera intrinsic. A similar problem can also be viewed on the examples of drill and can. The reconstructed segments can compensate for the inaccurate depth information and respect the object surface. On the other hand, the segmentation noise presented in the input segments of drill and can is also removed by the reconstruction.

As mentioned above, during testing, we calculated the Chamfer loss between the input and the reconstructed segments and the ADD and ADD-S loss. We then plot the histogram of the Chamfer loss with respect to the ADD and ADD-S loss, separately. Ideally, we expected a positive correlation between the Chamfer loss and the 6D pose loss, which means when the Chamfer loss is low, the 6D pose error is also low. If this is the case, we can use the Chamfer loss as a confidence measure of the 6D pose estimate during testing. However, as shown in Figure 5.7, this positive correlation can not be found in the LM dataset. The 6D pose error can vary in a large range at a very low Chamfer loss. Similarly, this positive correlation is also not found on the YCBV test set. Overall, the Chamfer loss between input and reconstructed segments can not be used as a confidence measure for 6D pose estimation. One possible reason is that the Chamfer loss is a point-to-point loss rather than a point-to-surface loss [2]. Since the reconstructed point cloud segments is expected to represent object surfaces, Chamfer loss can not measure how well the reconstruction adhere to the object surface. One possible improvement is to use an additionally point-to-surface quadric loss [2] for both training and testing the pose estimation network.



(a) ape

(b) drill

(c) can

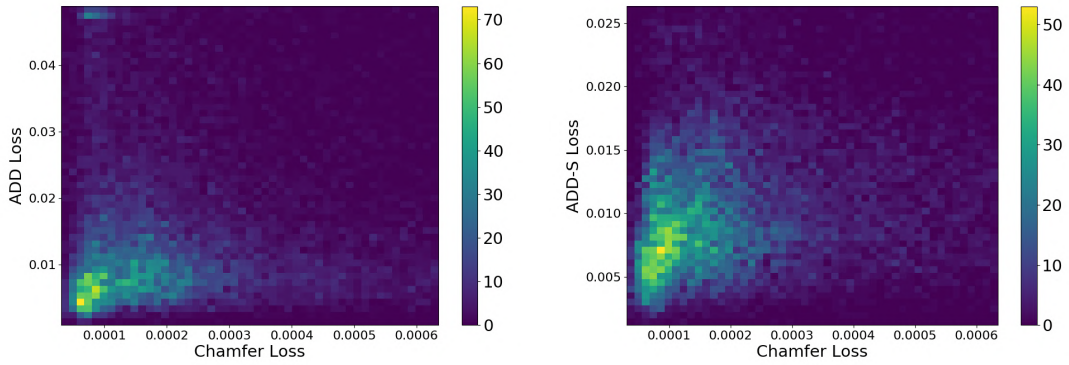
Figure 5.6: Examples of input segment (left column), reconstructed segment (middle column), and overlays of input and reconstructed segments (right column). The object model is shown in cyan wire frame. All the objects are from the LM dataset.

5.4.5 Time Performance

We measure the time performance on an Nvidia Titan X GPU. Our system is implemented with Tensorflow. Pose estimation by a forward pass through our network takes 0.07 seconds for a single object. The 10 iterations of ICP refinement require an additional 0.03 seconds.

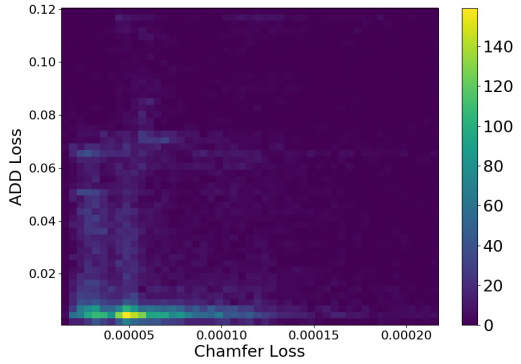
5.5 Summary

In this chapter, we proposed an augmented autoencoder based system CloudAAE for accurate and fast 6D pose estimation of known objects represented by point clouds. We use an augmented autoencoder as the base structure for the pose estimation system. We enforce the latent code to encode 6D object pose information

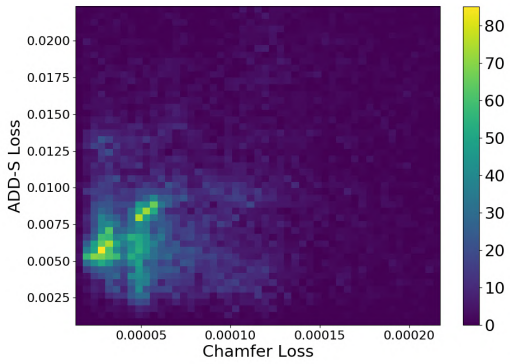


(a) Chamfer v.s. ADD (LM)

(b) Chamfer v.s. ADD-S (LM)



(c) Chamfer v.s. ADD (YCBV)



(d) Chamfer v.s. ADD-S (YCBV)

Figure 5.7: Histogram of Chamfer loss with respect to ADD (a, c) and ADD-S losses (b, d) on the LM and YCBV test set respectively.

and to ignore noise and occlusion. The latent code is used as the input to two pose regressors for regressing 3D rotation and 3D translation. Compared to CloudPose, CloudAAE reduces the sparsity of pose information from the input point clouds by learning a latent code encoding the pose information. Although CloudAAE does not reach state-of-the-art performance, the experiment result shows that it is more robust to noise in depth information.

As mentioned before, we focus on introducing the "autoencoder" aspect in this chapter. In the next chapter, we will investigate the "augmented" aspect. More specifically, we propose a data synthesis pipeline that enables us to add different data augmentation to the training data and study its impact on the system performance. Moreover, it is of interest to investigate whether the performance of CloudAAE can be improved by adding synthetic training data.

Chapter 6

CloudSyn: Online Data Synthesis with Point Clouds

In the previous two chapters, we investigated two methods for learning 6D object poses from point clouds. Compared to the previous two chapters, which focus on the problem of 6D pose estimation itself, we look into generating synthetic training data for learning 6D object poses in this chapter. The main reason is that deep learning-based methods have become the most popular method for tackling the problem of 6D object pose estimation. Furthermore, among numerous deep learning technologies, supervised learning is commonly used for this task [68, 130]. To enable training a deep learning-based system, a vast amount of annotated data is required. Commonly, there are two kinds of training data, namely, real training data and synthetic training data. We have already briefly touched on the difference between them and why researchers use synthetic data in Section 3.3. In this chapter, we dive into more details to motivate the need to use synthetic training data for the problem of 6D object pose estimation. We present a lightweight data synthesis pipeline that creates synthetic point cloud segments for training.

We motivate the use of synthetic training data and the general drawbacks of existing data sets in Section 6.1. Our data synthesis pipeline is presented in Section 6.2. Experimental results are shown in Section 6.3. Section 6.4 concludes this chapter.

6.1 The Whys

There are several motivations which will be introduced in this section. The motivation of synthetic data is describe in Section 6.1.1. The motivation of using point clouds as the data modality is presented in Section 6.1.2. The motivation of having an on-line data synthesis pipeline is presented in Section 6.1.3. Figure 6.1 shows an illustration of the practical aspects that motivate this work.

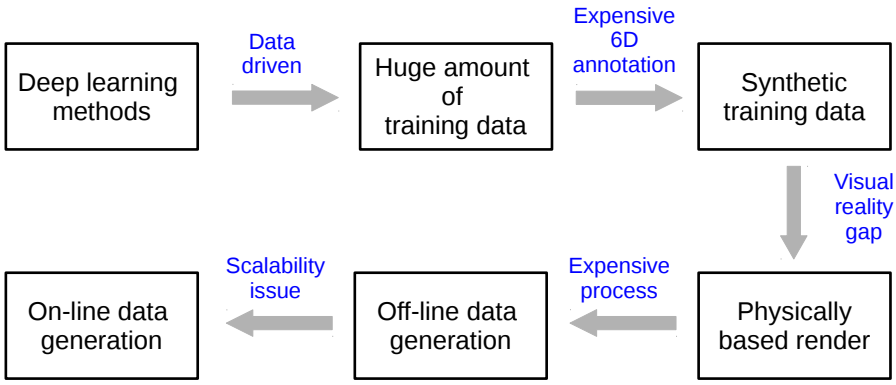
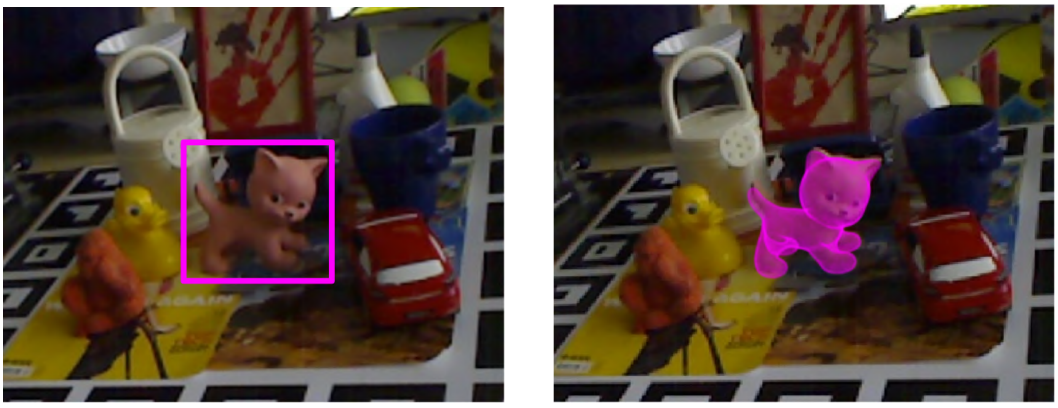


Figure 6.1: Overview of the practical aspects that motivate this work.

6.1.1 Why Synthetic Data?

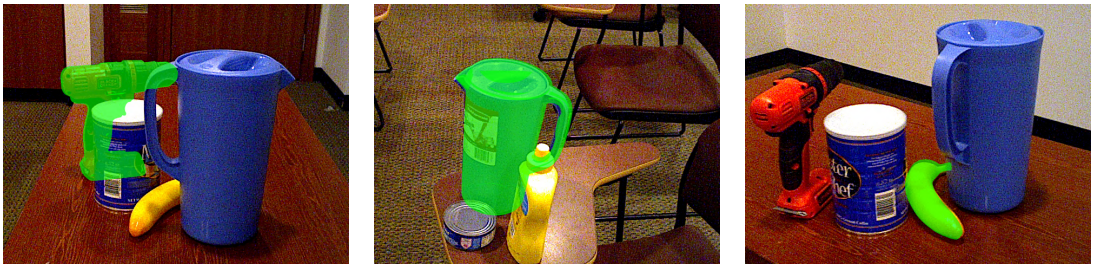
Deep learning-based methods are data-hungry. Therefore, to facilitate this requirement, the first common step is to collect a vast amount of training data from real-world setups. More importantly, those training data should be annotated according to the target applications. This means if the target application is 2D object detection, the corner and size information of 2D bounding boxes around the target objects should be annotated. Similarly, if the target application is 6D object pose estimation, the 3D translation and 3D rotation of target objects should be annotated. Figure 6.2 shows the illustration of those two annotations. The 6D pose is illustrated by first transforming the corresponding object model with the 6D pose and overlaying the transformed model onto the image for visual inspection. It can be inferred that compared to a 2D bounding box, the annotation for a metrically accurate 6D pose is very expensive. It takes a couple of seconds to annotate a 2D bounding box. On the other hand, it can take up to several minutes for annotating a 6D pose; with a sophisticated tool, the annotation time can be reduced to 30 seconds per object [88]. Furthermore, synthetic data can potential help to mitigate the dataset bias problem which we will elaborate on in Section 6.1.3.

As introduced before (Section 3.3), real datasets have been created for learning 6D object pose [139, 52]. Although those real-world datasets are very valuable contributions to the field, they have two main drawbacks. The first is the cost of data annotation as described above. The second is that sometimes the full accuracy of pose annotation can not be guaranteed. Figure 6.3 shows three examples of pose annotation, one accurate (a) and two slightly inaccurate (b,c) annotations. For the inaccurate annotations, it can be seen that the transformed object models in green do not fully cover the target objects. Since deep learning methods are data-driven, inaccurate pose annotations set an unsatisfying upper limit for model performance. In general, deep learning-based methods benefit from large amounts of high-quality training data. To this end, creating synthetic data not only eliminates the need for manual labeling but also can guarantee the accuracy of labels.



(a) 2D bounding box (b) 6D pose overlay

Figure 6.2: Illustration of 2D bounding box and 6D pose overlay. Images taken from LM dataset [50].



(a) Accurate annotation (b) Inaccurate annotation (c) Inaccurate annotation

Figure 6.3: Examples of 6D object pose annotations. Images taken from YCBV dataset [139].

6.1.2 Why Point Clouds?

Recalling from Section 3.3.2, there are three approaches for generating color-based synthetic training images: rendering with textured object models, "cut&paste", and physically-based rendering (PBR). Out of these three methods, PBR is the most computationally expensive approach because it aims to generate photo-realistic images. The motivation behind this is to fill the visual reality gap. The visual reality gap between real and synthetic RGB images is often large. This large gap can lead to the problem that systems trained on synthetic data can not generalize well to real-world testing data.

According to the recent benchmark challenge for 6D object pose estimation (BOP) [58], the performance of deep learning methods has improved compared to the previous year. The main reason for the performance boost of deep learning-based methods is to have additional photo-realistic synthetic images for training [58]. These images are created using PBR and present a much smaller visual reality gap than naive approaches such as "render & paste" [58]. Despite being effective, PBR is expensive in terms of time and hardware storage. This high cost

makes it challenging to scale PBR up to a large number of objects, which is often desired in robotic applications.

In contrast, the visual reality gap for depth information is much smaller, and methods using only depth information are more robust in the presence of the visual reality gap [57]. This indicates that if synthesizing data with only depth information, this gap is potentially much smaller and easier to fill. This opens up the possibility to create a lightweight data synthesis pipeline using only depth information presented in point clouds.

6.1.3 Why Online?

Due to the complicated and slow rendering process, PBR is often expensive time-wise. The most recent state-of-the-art BlenderProc [23] takes around one second to generate one photorealistic image containing $5-10$ objects. If one were to generate a dataset for training a network, it would take two days [23]. Furthermore, those datasets are often created offline due to the slow rendering [138]. This leads to another cost, which is the hardware storage cost. To have good coverage of the continuous pose space and illumination conditions, the resulting dataset is often in hundreds of gigabytes. It is costly storage wise to store all those synthetic data. Moreover, the overhead of writing and reading training data is also expensive time-wise.

There is one more concern that we referred to as the "dataset bias". As we have already described, either real or synthetic training data collection is not trivial. However, after a tremendous amount of effort, the resulting datasets often still cover a sparse range of the pose space. Figure 6.4 shows an illustration of this. We take both the training and testing 3D rotation pose annotations of the power drill in the YCBV dataset. The 3D rotations are converted to the axis-angle representation and plotted in a sphere with a radius of π. The training poses are in blue, and the testing ones are in red. Furthermore, we also converted the 3D rotations to the Euler angle format and plotted the corresponding histogram. It can be seen that the training data only covers a very sparse range of the 3D rotation space, and the testing data is in the same distribution as the training data. Hence, the testing data is bias to the training data. If we deploy a pose estimator trained on this data to a real-world robotic application setup, due to the sparse coverage, it is highly likely the real-world test poses are not covered by the dataset. This may lead to unpredictable behaviour of the pose estimator in the real world, although the performance on the testing dataset is good. It can also be seen from both the rotation sphere and the histogram that the dataset testing pose is only a small subset of the training poses. In general, perhaps the coverage is sufficient for training a method that performs well on a specific dataset. However, it is not practical in real-world scenarios since the desired pose distribution might lay outside of the training distribution.

A possible solution is to have a dataset that covers the full real-world pose space. However, this is theoretically possible but too impractical due to the cost of time and hardware storage. Furthermore, this may intensify the problem of pose

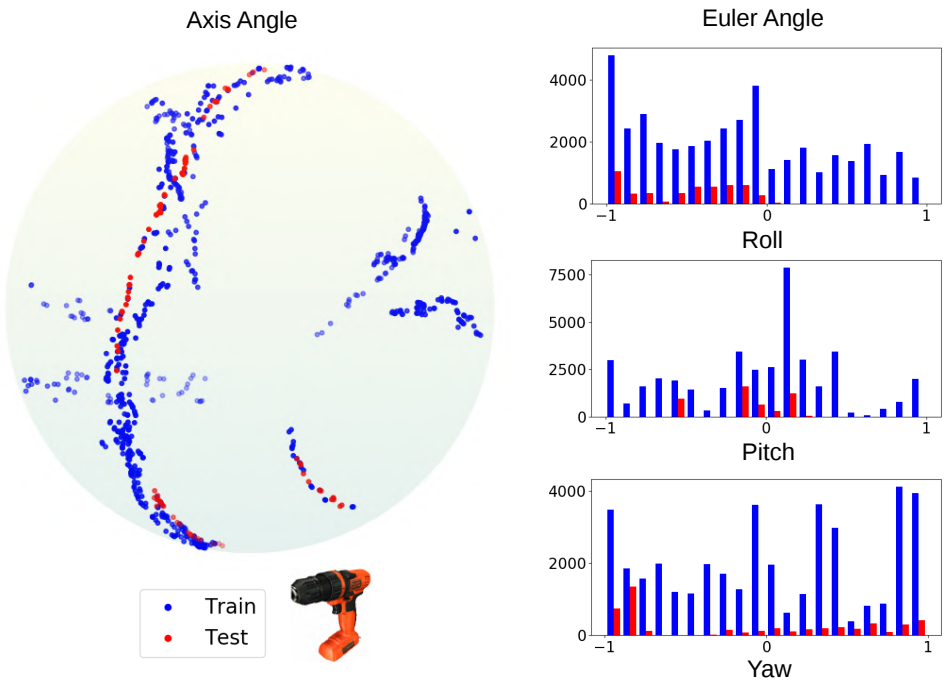


Figure 6.4: Illustration of the pose space coverage for one object in YCBV dataset.

ambiguity for rotational symmetric objects. A reasonable alternative solution to this is to have a lightweight on-line data synthesis process. Ideally, the synthesizing process is extremely fast to be afforded to use in an on-line manner. In this way, instead of having unconstrained pose ranges, the desired pose space can be provided for the data synthesizing accordingly. In other words, assuming you know approximately the pose space of your robot's workspace, you can provide this information to the data synthesis pipeline. In general, a cheaper data synthesizing process is very desirable for robotic applications, in which the desired pose distribution can vary from one workspace to another.

6.2 Data Synthesis with Point Clouds

According to the above-mentioned practical aspects, namely, having synthetic data, using point clouds, and having an on-line data synthesis process, we present a solution. Towards tackling those issues, we propose an on-line data synthesis system that requires a texture-less 3D object model and the desired viewpoint as the input. Moreover, the computational cost is low. Compared to existing approaches (Section 3.3.2), ours is computationally lightweight and does not need textured 3D object models or segmented foreground objects from real-world training data. Another difference is that the existing approaches mostly conduct off-line data generation while ours can be used on-line.

We show the effectiveness of our system on the LineMOD, Occluded LineMOD, and YCB Video datasets. Using only synthetic training data, our model achieves

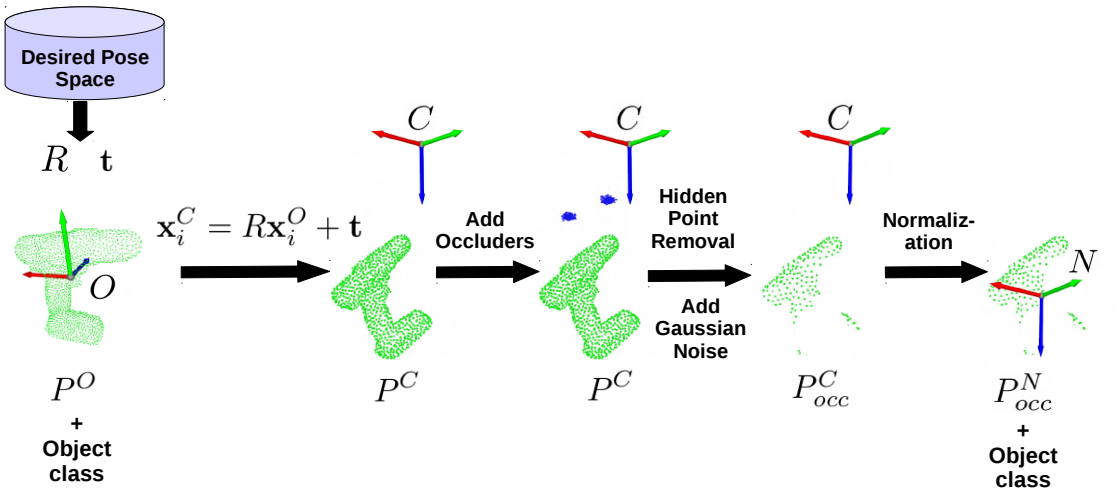


Figure 6.5: Data synthesis pipeline. The input for our on-line training pipeline is a 3D object model P^O with class information, and the desired 3D rotation R and translation $\mathbf{t}$. P^O is at object coordinate O. With R and $\mathbf{t}$, the object model is first transformed to the camera coordinate C. The transformed model is denoted by P^C. Spherical occluders S (denoted in blue) are added between C and P^C. Hidden point removal is applied to $P^C \cup S$ to remove points in P^C and are not visible from C. Zero-mean Gaussian noise is added to each remaining point to introduce variance. The final point set is P^C_{occ}. P^C_{occ} is normalized into P^N_{occ}, by subtracting the mean $\mu_{\mathbf{p}}$ of P^C_{occ}.

state-of-the-art performance among other synthetic trained methods on the LM dataset [52]. Our cheap synthetic point cloud data can replace costly render-based synthetic data for training systems using depth for pose inference. Our data synthesis process is up to three orders of magnitude faster than commonly applied approaches that render RGB image data. In general, we present a point cloud-based lightweight data synthesis pipeline for generating training data. Compared to existing RGB based data synthesis systems, the cost of ours is lower in the sense of time and hardware storage. We name our data pipeline as CloudSyn.

Figure 6.5 illustrates the data synthesis pipeline. The inputs for the data synthesis pipeline are a 3D object model and a 6D pose. We use 3D object models represented by point clouds. A 3D rotation R and a 3D translation $\mathbf{t}$ are drawn from the desired pose distribution. The desired pose distribution can be the poses in a training set [122, 98]. It can also be a pool of poses covering the same distribution as the poses in a training set [118, 137, 87]. In this work, if the training dataset contains a large number of poses, we use them as the 6D pose for data synthesis. Otherwise, we draw poses from the distribution of training poses.

6.2.1 System Architecture

Given an object model represented by a set of points $P^O = \{\mathbf{x}_i^O \in \mathbb{R}^3 \mid i = 1, 2, \ldots, n\}$, the model is transformed from its object coordinate system O to the camera coordinate system C by

$$\mathbf{x}_i^C = R\mathbf{x}_i^O + \mathbf{t}. \tag{6.1}$$

$P^C = \left\{\mathbf{x}_i^C \in \mathbb{R}^3 \middle| i = 1, 2, \ldots, n\right\}$ is the transformed model, where $\mathbf{x}_i^C$ is the ith point. To simulate occlusions, we randomly generate a set of points S as the spherical occluders (denoted in blue in Figure 6.5) between the camera origin C and P. Since we are interested in the problem of estimating the object pose from a single camera view, the desired sensor input is a single-viewed object segment rather than a full 3D model. To create a plausible object segment, we use the hidden point removal method proposed in [67]. We remove points in $P^C \cup S$ that are not visible from C by applying hidden point removal (HPR) [67]. From the set of visible points, we remove the points belonging to S. To add variance to the training sample, we add zero-mean Gaussian noise with standard deviation σ to each remaining point. In all of our experiments, we use $\sigma = 1.3mm$. The final resulting object segment is $P_{occ}^C = \{\mathbf{x}_i \in \mathbb{R}^3 \mid i = 1, \ldots, m\}$.

The data synthesis pipeline creates a point cloud segment P_{occ}^C, and P_{occ}^C is normalized by removing its coordinate mean $\mu_{\mathbf{p}}$ before further steps. We denote the normalized P_{occ}^C as P_{occ}^N. Using the data synthesis pipeline, we apply random noise and occlusion to the input point cloud segment to add variance in the training samples.

6.2.2 Spherical Occluder

To simulate external occlusions, we empirically generate two spherical occluders between the camera origin C and the target object P^C. The positions of each occluder are decided by the translation $\mathbf{t}$ and the view frustum of the camera. Figure 6.6 shows an illustration of view frustum. In general, the view frustum is defined by the near and far planes and the corresponding heights and widths. The distance from the occluder to the camera origin is randomly picked between the target object and the near plane. The occluder offsets along the x and y axes of the camera coordinate are randomly picked within the height and width of the near plane. Since the occluders are spherical, we do not consider a 3D rotation for them.

Once the center of the occluder is decided, we use the center as a mean to generate 200 3D points in a Gaussian distribution with a standard deviation of 0.01 m. Those 200 3D points together form a spherical occluder. Figure 6.7 shows some examples of the spherical occluders. For each example, we show two views of the target object and the spherical occluders. We also show the same views after hidden point removal and adding per point noise. The models of the target object are also presented for easier understanding. In the first two examples (Figure 6.7(a) and 6.7(b)), the occluders create good amount of occlusion for the target object. In

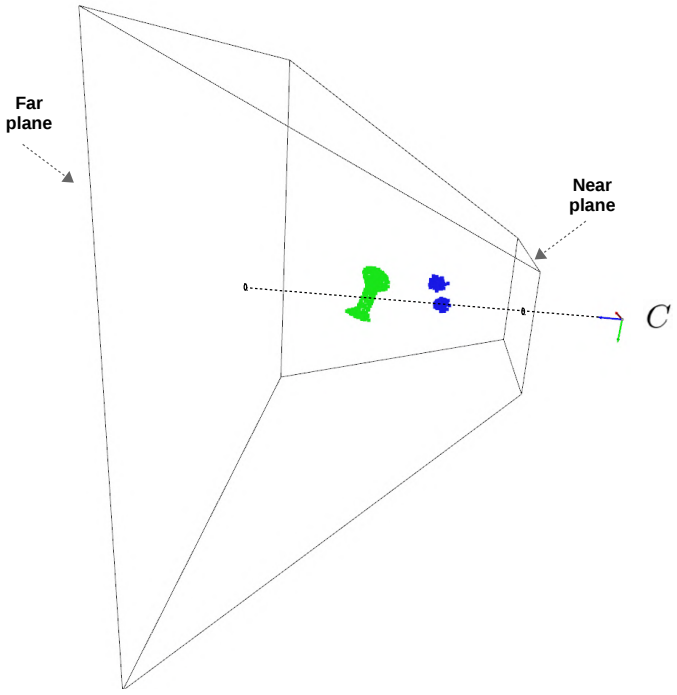


Figure 6.6: Illustration of the camera view frustum.

the third example (Figure 6.7(c)), the occluders only creates a very small amount of occlusion for the target object. In the last example (Figure 6.7(d)), the occluders do not create any occlusion and the resulting object segment is not occluded.

6.2.3 Hidden Point Removal

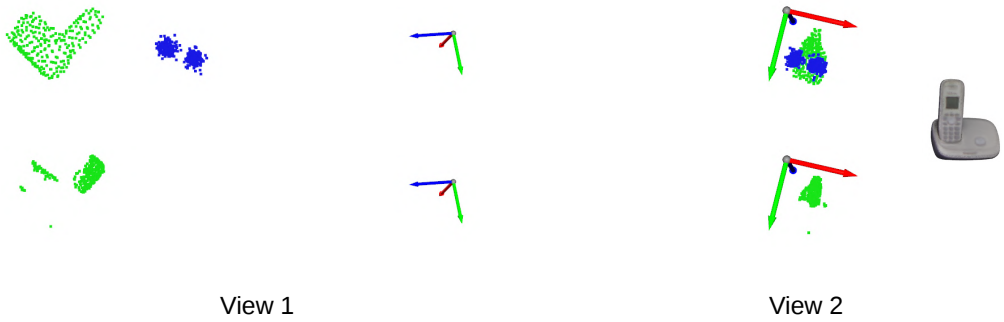
Given an object model represented by the set of points P and a viewpoint C, the HPR operator is used to determine the set of visible points $P_{vis} \in P$ if viewing P from C. The first step is to invert P. The inversion is achieved using the spherical flipping proposed in [66]. Assuming P is in a coordinate system in which C is the origin, and a 3D sphere with radius r is also centered at C. This sphere includes all points in P. The inversion $\hat{\mathbf{p}}_i$ of point $\mathbf{p}_i \in P$ is:

$$\hat{\mathbf{p}}_i = f(\mathbf{p}_i) = \mathbf{p}_i + 2(r - \|\mathbf{p}_i\|)\frac{\mathbf{p}_i}{\|\mathbf{p}_i\|}. \tag{6.2}$$

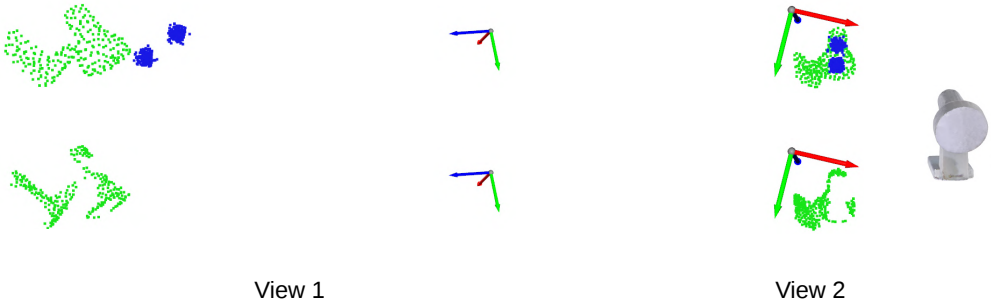
The second step is to construct a convex hull from the inverted points and the sphere origin. Let $\hat{P}$ be the inverted points: $\hat{P} = \{\hat{\mathbf{p}}_i = f(\mathbf{p}_i) \mid \mathbf{p}_i \in P\}$. A convex hull of the set $\hat{P} \cup \{C\}$ can be calculated. The points on the convex hull of $\hat{P} \cup \{C\}$ are the visible points P_{vis}.

6.3 Experiments

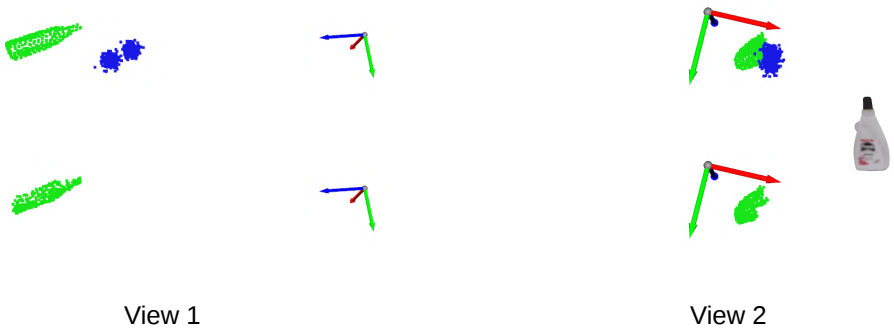
We train CloudAAE on the data generated by CloudSyn and compare our results with the state-of-the-art methods on public datasets, the LM [52], LMO [14] and



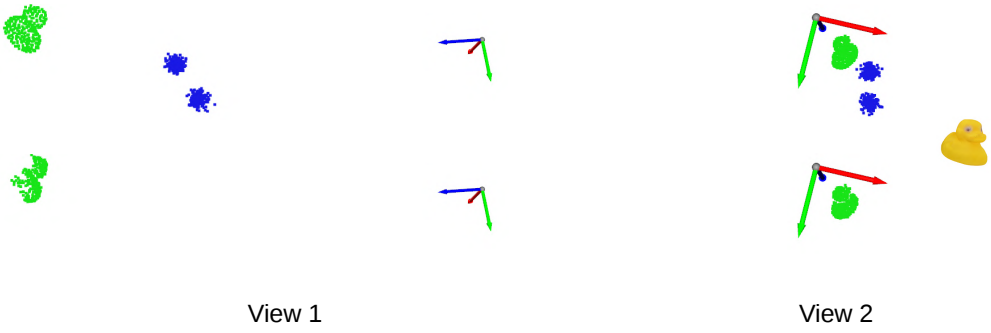
(a)



(b)



(c)



(d)

Figure 6.7: Examples of spherical occluders. Coordinate frame represents the camera. Target objects are in green dots and occluers are in blue dots. For each example, the positioning of occluders and the resulted object segments are shown in two views.

YCBV [139] datasets. In the ablation study, we investigate the impact of the amount of synthetic data used for training. We also compare our spherical occluder with another variant of occluders. We also investigate the impact of adding per point noise. Finally, we present the runtime and usage of hardware storage of our data synthesis pipeline.

6.3.1 Experiment Setup

Depending on the applicability of the methods, for each dataset, we compare to a subset of state-of-the-art methods SSD-6D [68], EEPG-AAE [134], CloudPose [36], PVNet [99], PoseCNN [139], DenseFusion [130], PVN3D [49] and PointVoteNet [42].

We use the 3D models in point clouds provided by the datasets for the on-line data synthesis. Tests are conducted on real test images from the official test split. LM provides approximately 200 training poses for each class, which is a small amount for generating synthetic data. To obtain more 6D poses for training, we first calculate a kernel density estimate (KDE) on the training set poses. Then we draw $100,000$ 6D poses from the distribution for each object class for data synthesis. For YCBV, we use the $80,000$ 6D poses from its synthetic training set as the 6D poses for data synthesis.

For training, we use the Adam optimizer with a learning rate of 0.0008. The batch size is 128. The number of points of the input point cloud segments is $n = 256$. Standard derivation for zero-mean Gaussian noise is 1.3 mm. We use the pipeline on both CloudPose and CloudAAE and report the results. For ICP refinement, we use the simple Point-to-Point registration provided by Open3D [151] and refine for 10 iterations. The initial search radius is 0.01 meter, and it is reduced by 10% after each iteration.

When training with synthetic data, both CloudPose and CloudAAE use the proposed data synthesis pipeline. For testing, our method, CloudPose, and DenseFusion [130] require an off-the-shelf semantic segmentation method. For LM and LMO, both our method and CloudPose use the test object segmentation provided by the corresponding dataset. For YCBV, ours, CloudPose, and DenseFusion use the object segmentation provided by PoseCNN [139]. When training with additional real data, we first generate the synthetic segments (P^O_{occ} in Figure 6.5) off-line and use them with the real data for training.

6.3.2 Comparison of Prediction Accuracy

We first compare the system performance of CloudAAE when using different training datasets. We then discuss our system performance when using only synthetic data for training on three datasets. We will also briefly discuss our system performance when using additional real training data.

CloudAAE with different training data We train CloudAAE with only real, only synthetic, and both real and synthetic data on all three datasets. The real

	Real	Synthetic	Both
LM	86.7	92.5	95.5
LMO	54.9	63.2	66.1
YCBV	-	93.5	93.6

Table 6.1: Performance of CloudAAE using different training data.

	RGB	D		RGBD
	EEGP-AAE [134]	CloudPose [36]	CloudAAE [35]	SSD-6D [68]
LM	89.2	75.2	92.5	90.9
LMO	-	44.2	63.2	-
YCBV	-	93.0	93.5	-

Table 6.2: Performance of methods training with only synthetic data. All results are with ICP refinement.

training data is directly from the corresponding datasets, and the synthetic training data is generated with CloudSyn. The test performance is shown in Table 6.1. It can be seen that with only CloudSyn synthetic training data, CloudAAE is able to achieve better performance compared with using real training data on LM and LMO. And the performances are further improved on all datasets when combining the real and synthetic training data. This shows the usefulness of CloudSyn synthetic data as the training data.

Synthetic training data only Table 6.2 shows the evaluation results on three datasets. All results are with ICP refinement. On LM, we achieve state-of-the-art performance using only depth for pose inference. This shows the effectiveness of the combination of our data synthesis pipeline and CloudAAE. Both CloudAAE and CloudPose use depth for pose inference, but our method generalizes better to real test data. This shows learning a latent code that encodes pose information can improve system robustness. Using depth for pose inference, our method outperforms the RGB-D based method SSD-6D.

On LMO, CloudAAE outperforms CloudPose by a large margin. Among the methods without ICP, ours has a better performance than PVNet and PointVoteNet, which use real data for training. This shows that our system can also handle occlusion. On YCBV, CloudAAE is slightly better compared to CloudPose. Moreover, although trained on synthetic data only, our method has comparable performance to other methods trained on both real and synthetic data. To the best of our knowledge, we are the first to report the results using only synthetic training data on this dataset.

	RGB[1]		D		RGBD		
	PoseCNN [139]	PVNet [99]	PointVotNet [42]	CloudAAE [35]	PoseCNN [139]	DenseFusion Iterative [131]	PVN3D [49]
LM	-	-	-	95.5	-	-	99.4
LMO	24.9	47.3	52.6	66.1	78.0	-	-
YCBV	-	-	-	93.6	-	93.2	96.1

Table 6.3: Performance of methods training with both synthetic and real data. All results are with ICP refinement except for the RGB based methods.

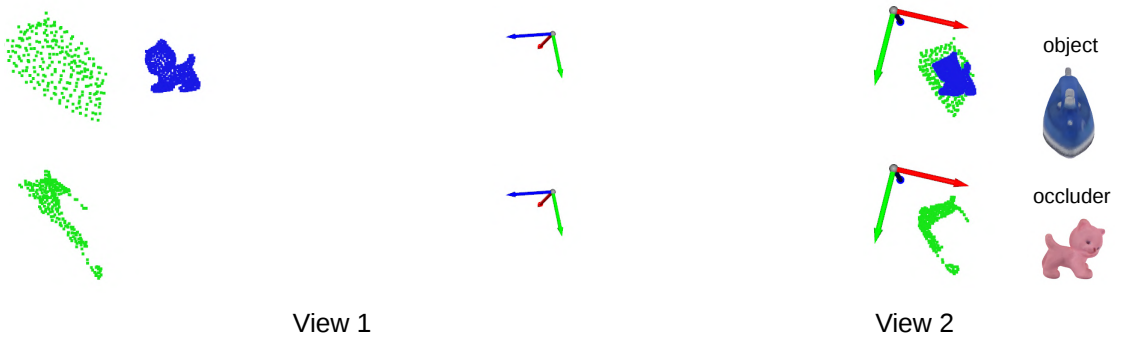
Synthetic and real training data As shown in Table 6.3, using additional real training data boosts the performance of CloudAAE on all three datasets. On LM, although CloudAAE with ICP has worse performance compared to RGB-D based PVN3D, we still obtained promising performance using only depth. On LMO, among the methods without ICP, ours has the best performance. With ICP, PoseCNN outperforms our method. One possible reason is that PoseCNN uses a sophisticated ICP while we use a standard point to point ICP. Moreover, it shows LMO is very challenging for methods that rely on a single modality for pose inference, and combining both color and depth is very beneficial. Another possible factor limiting our performance is that there might be a visual reality gap between our occlusion simulation and the real data. On YCBV, we report the results when using real training data with synthetic data from CloudSyn. Using real and the two kinds of synthetic data achieves similar performances, and ours is comparable with the RGB-D methods [130, 49].

6.3.3 Ablation Study

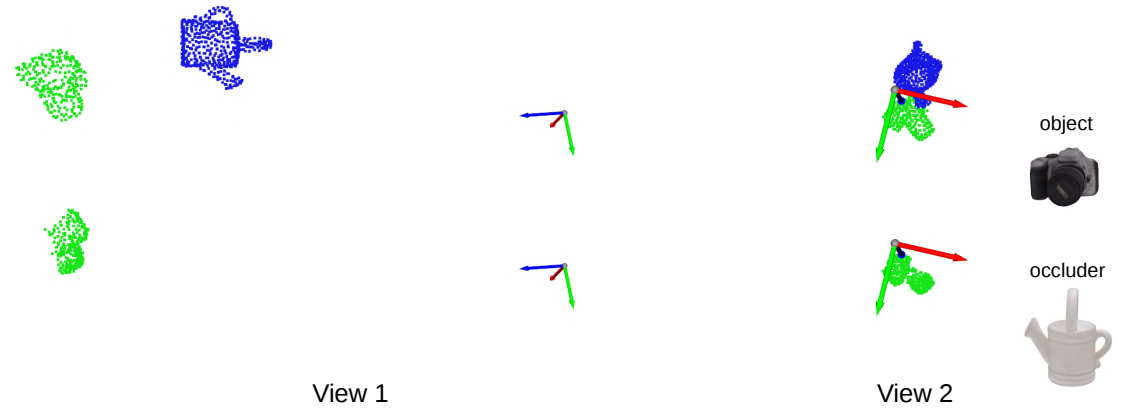
Different occluders We investigate the impact of spherical occluders as well as another different strategy of simulating occlusions. For a target object, instead of generating random spherical occluders, we pick a random object from the LM objects as the occluding object. Using an object occluder, we try to simulate the target object is occluded by another object, as is usually the case in the LM dataset. The 3D translation selection for this occluder is the same as described in Section 6.2.2. A random 3D rotation is generated for this occluder. Figure 6.8 shows examples of object occluders.

We train one network without occluders and one network with object occluders, and compare the performance with spherical occluders. The result is shown in Table 6.4. It can be seen that without occluders, the test performance presents a huge drop. Hence, the occluders are necessary to achieve good testing results. Using an object occluder makes the performance slightly worse. One possible reason is that it is difficult to pick the optimal positions for the object occluder. Because the object occluder is significantly larger than the spherical occluder, we only use one object occluder for synthesizing each sample. On the other hand, we use

[1]Results are without ICP refinement



(a)



(b)

Figure 6.8: Illustration of random object occluder.

occluder	No	Object	Spherical
avg	73.0	80.6	82.1

Table 6.4: Results (w/o ICP) with different occluders on LM.

two spherical occluders for synthesizing each sample. Since the positions of each spherical occluder are picked separately, we potentially have a higher overall chance for more optimal positioning of the occluders. On the other hand, there are some overhead computations for choosing the object occluder and its 6D poses, and the computational cost is higher than using the spherical occluder.

Per point noise We investigate whether the per point noise has an impact on the system performance. We remove the step of adding per point Gaussian noise and train a network. The performance is shown in Table 6.5. Removing the per point Gaussian noise during the training phase leads to surprisingly bad test performance. For more detailed insight, we also show the individual results for each object.

	ape	bvise	cam	can	cat	driller	duck	e.box	glue	holep	iron	lamp	phone	avg
w\o noise	30.8	83	30.1	75	31.8	92.2	8.8	99.7	88.1	22.5	63	79	71.5	59.7
w. noise	74.5	86.6	65.6	90.2	90.7	97.3	50	99.7	93.5	57.9	85	82.1	94.4	82.1

Table 6.5: Results (w/o ICP) with and without per point noise on LM.

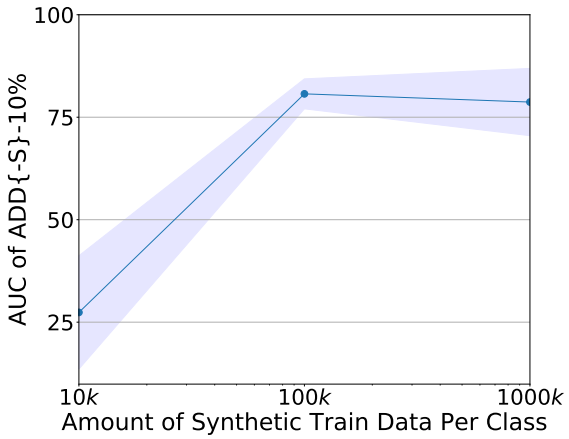


Figure 6.9: Amount of synthetic training data and performance accuracy (w/o ICP).

Recalling the object models presented in Figure 2.9(a), it can be observed that the performance has dropped significantly for small objects such as ape, cam, cat, duck, and hole puncher. For bigger objects such as the driller and lamp, the performance is slightly degraded. One possible reason is that due to both the data noise and inaccurate depth in this data set, in general the surface shape of all objects is distorted. Smaller objects might be impacted more by the distortion, compared to the bigger objects. In this case, adding some noise to the synthetic data during training can help the network to generalize better during testing with real data.

Amount of training data We study how the amount of synthetic training data impacts the system performance. For each object class in LM, we generate either $10,000$, $100,000$, $1,000,000$ training poses per class. For each pose data amount, we train multiple networks with early stopping. If the training error of the current epoch failed to improve more than 10% of the previous lowest training error, the training process is terminated. We conducted five trials for $10,000$, and three trials for $100,000$ and $,1000,000$. We generate a set of training data for each trial separately. The trained networks are evaluated on the test set of LM. Figure 6.9 shows the test accuracy (without ICP). $100,000$ samples per class are sufficient for the network to perform well on the test set, and using $1,000,000$ per class does not provide further performance gain.

6.3.4 Runtime and Hardware Storage

We measure the time performance on an Nvidia Titan X GPU. Our system is implemented with Tensorflow. For the on-line data synthesis, it takes under 30 milliseconds to generate 128 object segments. The rendering-based approach [23] takes 1-4 seconds to generate an image with 10-20 objects. In the LM experiments, the 3D object models and $13 \times 100,000$ training poses take 128 MB of storage. If to generate PBR images [58] with similar number of objects (e.g. 10 objects per image), $13 \times 10,000$ images would take 63 GB.

6.4 Summary

In this chapter, we present a point cloud-based lightweight data synthesis pipeline. The data synthesis pipeline requires texture-less 3D object models and 6D object poses as the input. The 3D object models are provided by the corresponding dataset, and the 6D poses can be drawn from desired 6D pose spaces. The data synthesis pipeline is low cost in terms of both time and hardware storage. It takes ~ 0.2 milliseconds to create one training sample and can be used on-line. Using the synthetic data created by our data synthesis pipeline, our pose estimation system achieves state-of-the-art performance among other synthetic trained methods on a public benchmark. Moreover, our cheap synthetic point cloud data can replace expensive render based synthetic data for training systems using depth for pose inference. Our lightweight data synthesis pipeline enables more agile deployment of object pose estimation systems in robotic applications.

In Chapter 8, we will test this data pipeline in a real-world setup. Specifically, we opt for an in-hand object pose estimation task. Since our data pipeline is very flexible and each component can be replaced according to the application, it will be interesting to test how easy it is to adapt to a slightly different task. Also, the in-hand object pose estimation is very useful for applications such as human-robot hand-over.

Chapter 7

SSV: Saliency-guided Adaptive Seeding for Supervoxel Segmentation

In the previous chapters, we assumed that the object segment is available, and our pose estimation system uses the target object segment as inputs. In this chapter, we switch gears and investigate a problem in the field of segmentation. Specifically, we investigate the problem of over-segmentation on point clouds. The result of over-segmentation on 2D images is referred to as "superpixel" [106], while its counterpart on point clouds is called "supervoxel". Over-segmentation is a popular way of grouping pixels or voxels to larger entities and thus to strongly reduce the number of primitives that subsequent modules have to deal with [117]. This is especially important in robotics scenarios, where real-time constraints usually make the full interpretation of high-resolution image or point cloud data unfeasible.

Many applications use superpixels or supervoxels as the input. For example, in object discovery, the superpixels are grouped to obtain object candidates, which in turn are used as input for object recognition methods [61, 65, 39, 135]. Other applications include automatic object handle grasping [116], unknown object manipulation in cluttered environments [12], and semantic segmentation [136]. Since superpixels serve as input to all further processing, their quality has a significant impact on the quality of the output [46]. For example, violating object boundaries will introduce a permanent error into the processing pipeline since all following algorithms will be forced to use superpixels that contain more than one object [97]. In other words, if a single superpixel contains pixels belonging to two distinct objects, the two objects can not be fully distinguished in any subsequent processing step. The quality of a set of superpixels can be assessed based on, e.g., their adherence to object boundaries, compactness, smoothness, and a controllable number of superpixels [117]. Among these qualities, boundary adherence is one of the most critical requirements due to the reasons mentioned above.

In this chapter, we propose a saliency-guided method for generating supervoxels in 3D space. Rather than using an evenly distributed spatial seeding procedure, our method uses visual saliency to guide the process of supervoxel generation. This re-

sults in densely distributed, small, and precise supervoxels in salient regions which often contain objects and larger supervoxels in less salient regions that often correspond to the background. Our approach vastly improves the quality of the resulting supervoxel segmentation in terms of boundary recall and under-segmentation error on publicly available benchmarks. We explain the motivation behind using visual saliency for over-segmentation in Section 7.1. The background and related work on over-segmentation are presented in Section 7.2. In Section 7.3, we give an overview of our proposed method and details of each step. We evaluate our method in Section 7.4 on two publicly available datasets. Section 7.5 gives concluding remarks.

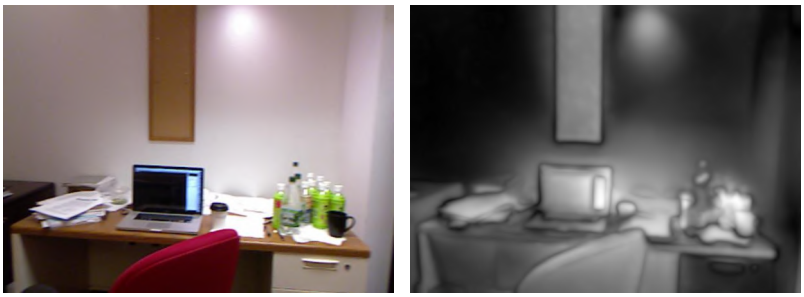
7.1 Why Over-segmentation with Visual Saliency?

As mentioned above, we are interested in generating supervoxels from point clouds. While dozens of superpixel algorithms exist that operate on 2D color images [1, 30, 144, 124, 142], methods that analyze RGB-D data and generate so-called supervoxels are much less common [97, 145]. Among the few methods that exploit depth, the Voxel Cloud Connectivity Segmentation (VCCS) supervoxels presented by Papon et al. [97] is probably the best known and most used method[1]. VCCS oversegments a 3D point cloud into supervoxels by applying a spatial seeding procedure that places the initial supervoxel centroids on a regular grid. The main difficulty of the method, as well as of other superpixel and supervoxel methods, is to find an appropriate parameter setting that defines a good trade-off between the number of supervoxels and the precision. On the one hand, obtaining a low number of large superpixels is favorable since this reduces the computational burden of subsequent processing. On the other hand, the superpixels should not be larger than the smallest objects, otherwise these will not be segmented appropriately and will be permanently lost in the further process [1].

An alternative to the uniform seeding strategy is to place the initial seeds adaptively. Moreover, the adaptive seeding process should follow certain priors. We propose to use visual saliency as the prior. The idea is motivated by the fact that saliency highlights regions that visually stick out of the image, i.e., they differ according to some features — such as color, intensity, or depth — from their background [32]. Figure 7.1(a) shows an original image and its corresponding color-based saliency map is shown in Figure 7.1(b). It can be seen that objects such as the laptop computer and the wooden board on the wall are highlighted in the saliency map. We argue that visual saliency is also a good indicator for the presence of objects since objects also usually differ from their surroundings [39]. This property is especially useful since it enables us to initialize a higher density of seeds in regions that probably correspond to objects.

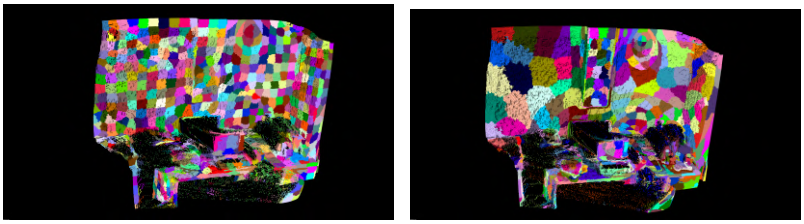
We propose a supervoxel algorithm that, instead of seeding the initial supervoxel centroids evenly spaced over the whole input data, applies saliency-guided seeding as illustrated in Figure 7.1. To achieve this, we cluster the input data according to visual saliency, assigning a saliency-specific seeding resolution to each

[1]The code for VCCS is publicly available as part of the Point Cloud Library (PCL).



(a) Original image (b) Saliency map

(c) Superpixel (VCCS) (d) Superpixel (SSV)



(e) Supervoxel (VCCS) (f) Supervoxel (SSV)

Figure 7.1: Visualization of supervoxel computation in a uniform manner and with our saliency-guided adaptive seeding method SSV. Top: RGB image (a) and corresponding saliency map (b). Middle: 2D projection of supervoxels of uniformly distributed supervoxels from VCCS [97] (c) and of our SSV method (d). Bottom: 3D supervoxel clouds for VCCS (e) and SSV (f).

cluster. Highly salient regions will have a greater density of seeds and vice versa. This leads to small, precise supervoxels in salient regions, and to large supervoxels in less salient ones, for example, on homogeneous background surfaces (see, e.g. the wall in Figure 7.1(f)). We show that, with the same average number of supervoxels per image as VCCS, we get a clearly improved boundary recall, undersegmentation error, and explained variation of the supervoxel segmentation. We thus show that visual saliency provides a useful prior that allows the segmentation to better respect object boundaries.

7.2 Related Work in Over-segmentation

Following the classification of Stutz et al. [117], superpixel methods can be classified into watershed-based [86], density-based [127], graph-based [30], contour evolution [79], path-based [33], clustering-based [1, 133, 124] and energy optimization methods [125, 144]. Recently, some deep learning-based methods were also proposed [64, 142]. Among the most popular methods is the Simple Linear Iterative Clustering (SLIC) superpixel method [1] that applies local k-means clustering of the image pixels on a regular grid pattern over the entire image to generate perceptually uniform superpixels. SLIC is appealing since it is simple, fast, and has only two parameters that must be controlled: the number of superpixels that shall be generated and the compactness of the superpixels. Many other superpixel and supervoxel methods follow the idea of SLIC, e.g., [97, 133], and [143].

Few methods integrate depth data into the segmentation process. The Depth-Adaptive Superpixels (DASP) [133] extend the iterative local clustering approach by introducing the control of superpixel seed density based on depth information. Similarly, Yang et al. [143] enhance the superpixel generation process using the depth difference between pixels to prevent violating boundaries between objects of similar color. They also use a local k-means clustering approach with an eight-dimensional distance measure including color, 2D, and 3D coordinates. Both methods can be classified as 2.5D since they use depth to improve superpixel generation but do not create 3D supervoxels but rather 2D superpixels. There are also works that exploit the 3D geometry of the scene and generate a full 3D supervoxel graph from point clouds [97, 140, 76]. Voxel Cloud Connectivity Segmentation [97] generates supervoxels from RGB-D data while guaranteeing that all voxels within each supervoxel are spatially connected. Similar to SLIC and DASP, VCCS is also a variant of iterative local clustering applied on a regular lattice. Xiao et al. [140] formulate over-segmentation as an energy minimization problem and propose a merge-swap optimization framework. Recently, Landrieu et al. [76] formulate this problem as a supervised learning problem. They proposed a graph-structured contrastive loss and used it in a metric learning framework.

Our Saliency-guided Supervoxel (SSV) method is based on VCCS due to its simplicity. Instead of uniformly distributing the supervoxels over the data, we use an adaptive seeding procedure guided by saliency. Our results show that our method clearly improves the quality of the supervoxels according to several evaluation metrics. It should be mentioned that in the literature, other methods are also referred to as supervoxels, which use time instead of space as the third dimension and are thus a type of video segmentation [33, 141]. These methods do not apply to RGB-D point clouds since they usually require a solid 3D volume of time and space as input. Thus, such methods are not comparable with the supervoxel methods operating on 3D data from a sensor such as the Microsoft Kinect.

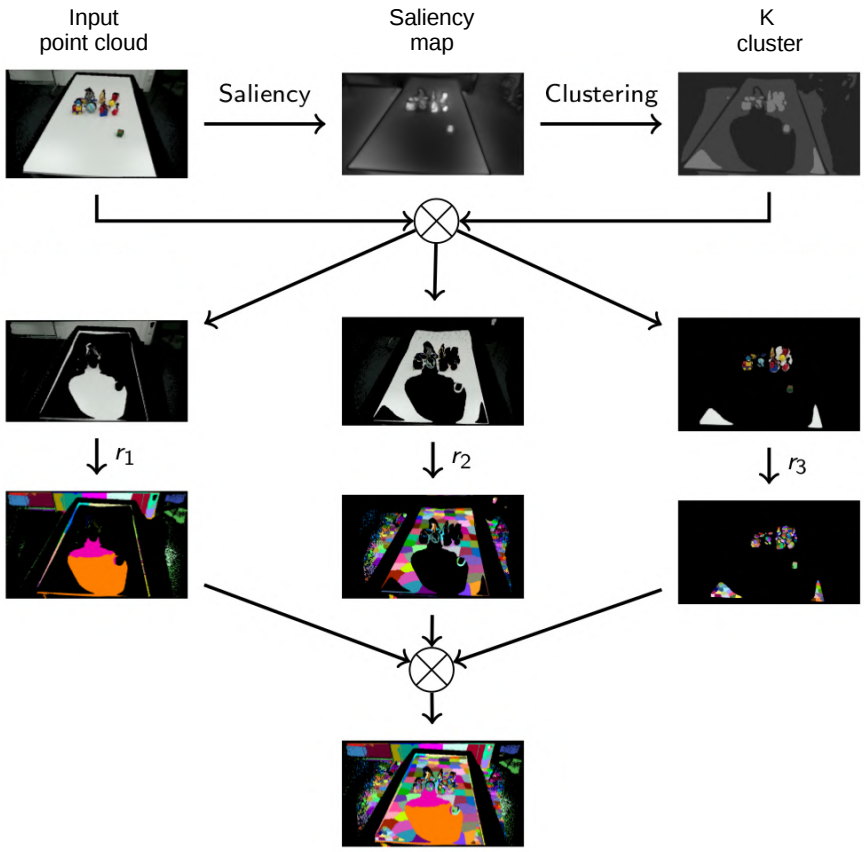


Figure 7.2: System overview. First, the saliency system VOCUS2 [32] generates a saliency map. The point cloud is partitioned into K clusters using k-means on the saliency values, and each cluster k is assigned a seeding resolution r_k. The VCCS supervoxel segmentation method [97] is applied to each cluster independently, and the results are combined to form the final output.

7.3 Saliency-guided Supervoxel Segmentation

In this section, we present Saliency-guided Supervoxels (SSV), our algorithm for generating supervoxels using saliency-guided adaptive seeding. SSV combines a visual saliency model, a newly introduced saliency-guided adaptive seeding approach, and a supervoxel segmentation algorithm.

An overview of our algorithm is presented in Section 7.3.1. More details on the visual saliency model and supervoxel computation are given in Sections 7.3.2 and 7.3.3, respectively. The adaptive seeding approach is described in Section 7.3.4.

7.3.1 System Overview

As illustrated in Figure 7.2, the input to SSV is an RGB-D point cloud. During pre-processing, the RGB information is passed to a visual saliency system, here VOCUS2 [32], to generate a pixel-level saliency map. The pixel-level saliency map is segmented into K regions applying k-means. The same segmentation is applied

to partition the point cloud into K clusters. The clusters are sorted in an ascending order based on their average saliency values.

The supervoxel seed resolution is determined according to the average saliency value: clusters with a higher average saliency have denser seeding. As the supervoxel method, we use VCCS [97], which is applied to the points in each cluster independently. The results are merged to obtain the final supervoxel segmentation. As setting $K = 1$ reduces our method to be equivalent to VCCS, SSV may be viewed as a generalization of VCCS.

7.3.2 Saliency Model

To compute a pixel-level saliency map from the RGB data, we use the computational visual saliency method VOCUS2 [32]. VOCUS2 converts its input image into an opponent-color space with intensity, red-green, and blue-yellow color channels. Center-surround contrasts are then computed on multiple scales by Difference-of-Gaussian filters operating on center and surround twin pyramids. Finally, the saliency map is generated by fusing the contrast maps across scales and channels using a simple arithmetic mean.

We use VOCUS2 since it is fast and has obtained good results on several benchmarks. Furthermore, it does not have a center-bias, which is essential in robotic applications since objects of interest are usually not in the center of the image. Despite these considerations, any computational visual saliency model could be applied in SSV.

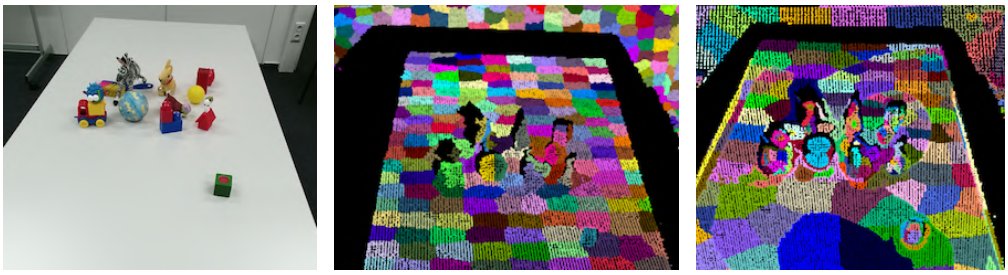
7.3.3 Voxel Cloud Connectivity Segmentation

VCCS is a supervoxel segmentation algorithm introduced by Papon et al. [97]. It first voxelizes the input RGB-D point cloud by equally partitioning the 3D space using an octree structure. The size of each voxel is defined by the voxel resolution R_{voxel}. Supervoxel seeds are generated uniformly on a regular grid with resolution R_{seed} that is much larger than R_{voxel}. Higher R_{seed} results in fewer supervoxels and vice versa. For each occupied seed voxel, the nearest cloud voxel is selected as an initial seed. Unoccupied seed voxels are discarded.

After selecting the initial seeds, a local clustering of voxels is performed iteratively until all voxels are assigned to supervoxels. The clustering is performed in a 39 dimensional feature space $\mathbf{F} = [x, y, z, L, a, b, \mathrm{FPFH}_{1\ldots33}]$, where x, y, z are spatial coordinates, L, a, b denote color in CIELab space, and $\mathrm{FPFH}_{1\ldots33}$ are the 33 elements of Fast Point Feature Histogram [110]. Each voxel is assigned to the supervoxel to whose centroid it has the smallest normalized distance

$$D = \sqrt{\frac{\lambda {D_c}^2}{m^2} + \frac{\mu {D_s}^2}{3{R_{seed}}^2} + \epsilon {D_{\mathrm{HiK}}}^2}, \tag{7.1}$$

where D_c is the Euclidean distance in CIELab color space, D_s denotes the spatial distance, D_{HiK} is the Histogram Intersection Kernel of FPFH [6], m is a nor-



(a) Input image (b) Supvervoxel (VCCS) (c) Supervoxel (SSV)

Figure 7.3: Input image (a) and supervoxels obtained with VCCS (b) and with our SSV method (c).

malization constant, and λ, μ and ϵ are the weights for each distance such that $\lambda + \mu + \epsilon = 1$.

7.3.4 Saliency-guided Adaptive Seeding

For our saliency-guided supervoxel seeding, we first compute a saliency map as described above. We then use k-means to partition the pixel-level saliency map into K clusters. The clusters are then sorted in ascending order according to the average saliency of the pixels in each cluster.

To control the size of supervoxels, the minimum and maximum seed resolution R_{min} and R_{max}, respectively, are defined. The seed resolution r_k for the k-th cluster is

$$r_k = 10^{\log R_{max} - (k-1)d}, \tag{7.2}$$

where the step size d is determined by

$$d = -\frac{\log R_{min} - \log R_{max}}{K - 1}. \tag{7.3}$$

By Equation (7.2), regions with high saliency are seeded densely, while less salient regions have a sparser seed distribution.

We apply VCCS (Section 7.3.3) independently to each cluster, using r_k as the seed resolution R_{seed} for data in cluster k. The final result is obtained by combining the K supervoxelizations into a single multiple seed resolution supervoxel representation. Figure 7.3 shows an example supervoxelization, with the VCCS result shown for comparison.

7.4 Experimental Results and Evaluation

We compare SSV against the current state-of-the-art supervoxel method VCCS [97]. Although some 2.5D methods such as DASP [133] could be adapted for the comparison, we chose to compare only to VCCS since it shares with SSV the key

property that the generated supervoxels are guaranteed to be spatially connected. This is important if the result is to be applied, e.g., for robotic manipulation. The evaluation procedure is described in Sec. 7.4.1, the datasets in Section 7.4.2, and the experimental results in Section 7.4.3.

7.4.1 Evaluation Metrics

We compare SSV and VCCS using the superpixel benchmarking tool[2] of Stutz et al. [117]. The tool provides a platform to evaluate the performance of superpixel algorithms using common metrics such as boundary recall (REC), undersegmentation error (UE), and explained variation (EV). The benchmark tool evaluates supervoxel methods by projecting each supervoxel back to the 2D image plane and then evaluating the result similarly as superpixel methods. This is reasonable since most publicly available datasets provide the ground truth only as a 2D image.

The notation we use in this section is as follows. Given an input image $I = \{x_j\}_{j=1}^{J}$ with J pixels, we write $S = \{S_n\}_{n=1}^{N}$ to denote a segmentation of I into N superpixels, and $G = \{G_m\}_{m=1}^{M}$ to denote the M ground truth segments.

Boundary recall (REC) [89] assesses boundary adherence by measuring the percentage of the superpixel edges that fall within a certain range of an arbitrary ground truth boundary. The range is defined as $(2r+1)^2$, with $r = 0.0025 \times L$ where L is the image diagonal size [117]. Higher boundary recall indicates a better boundary adherence.

Undersegmentation error (UE) [79] measures the amount of "leakage" of a segmentation S with respect to the ground truth G:

$$\mathrm{UE}(S,G) = \frac{1}{M}\sum_{m=1}^{M}\frac{\left[\sum_{\{n|S_n\cap G_m\neq\emptyset\}}|S_n|\right]-|G_m|}{|G_m|}, \tag{7.4}$$

where the leakage of superpixel S_n with respect to ground truth G_m is represented by the inner term. Lower undersegmentation error demonstrates less leakage.

Explained variation (EV) [92] attempts to measure the quality of a superpixel segmentation without relying on human-annotated ground truth. EV is defined as

$$\mathrm{EV}(S) = \frac{\sum_{n=1}^{N}|S_n|\left(\mu(S_n)-\mu(I)\right)^2}{\sum_{j=1}^{J}(x_j-\mu(I))^2}, \tag{7.5}$$

where x_j is the value for pixel j, $\mu(I)$ is the global pixel mean and $\mu(S_n)$ is the mean value in superpixel n. EV is the proportion of variation that can be explained by superpixel segments. A higher EV indicates better performance.

[2] https://github.com/davidstutz/superpixel-benchmark

7.4.2 Datasets

We evaluate on the NYUV2 [114] and SUNRGBD [115] datasets. NYUV2 contains 1449 color images with associated depth information. The data are collected applying Microsoft Kinect v1 and depict varying indoor scenes. We use the test set of 399 randomly chosen images specified in the superpixel benchmark to evaluate our method. We tuned the parameters of our method on a disjoint set of training images from this dataset.

SUNRGBD has 10335 images of cluttered indoor scenes, with color and depth information. The dataset contains images collected applying the Intel RealSense, Asus Xtion, Microsoft Kinect v1, and v2 sensors. We use 400 randomly chosen images from this dataset to evaluate our method.

7.4.3 Experimental Results

We varied the seed resolution R_{seed} between 0.05 and 0.25 for VCCS. For SSV, we created between 2 and 6 clusters, with minimum seed resolution R_{min} between 0.05 and 0.25 and maximum seed resolution R_{max} between 0.2 and 0.4. The values were chosen to obtain approximately the same average number of superpixels for both methods[3]. For both methods, we set $R_{voxel} = 0.02$, color weight $\lambda = 0.7$, spatial distance weight $\mu = 0.15$ and HiK distance weight $\epsilon = 0.15$ (Equation (7.1)).

Figure 7.4 shows the REC, UE, and EV for SSV and VCCS as a function of the average number of superpixels. The data in the figure correspond to the seed resolution R_{seed} for VCCS between 0.1 and 0.2. For SSV, we set $K = 6$, R_{min} between 0.05 and 0.2, and $R_{max} = 0.3$. The means and their 95% confidence intervals are plotted. The figure shows that with the same amount of superpixels, SSV performs significantly better in terms of REC and better or the same in terms of UE and EV. This is especially true for smaller numbers of superpixels (left parts of plots), which is of special interest for many applications that require low complexity. Due to the saliency prior, supervoxels generated by SSV are more efficient: a larger fraction of the supervoxels are in regions with many boundaries to preserve, while fewer supervoxel are used in background regions.

Table 7.1 indicates how the performance of VCCS and SSV varies as a function of the key parameters: R_{seed} for VCCS, and K, R_{min}, and R_{max} for SSV. We note an expected trend that a higher number of superpixels leads to higher boundary recall and explained variation, as well as lower undersegmentation error. However, more superpixels also correspond to less reduction in the complexity of the input representation.

In Figure 7.5, we show a qualitative comparison of our method and VCCS. The fact that SSV uses saliency to guide the seeding process is seen from having a higher density of supervoxels in highly salient areas that are likely to contain objects. Supervoxels are generated with a low density in non-salient background areas such as walls, resulting in larger supervoxels. We also see that small objects

[3]For both methods, it is not possible to determine the number of superpixels precisely in advance, due to the projection from 3D.

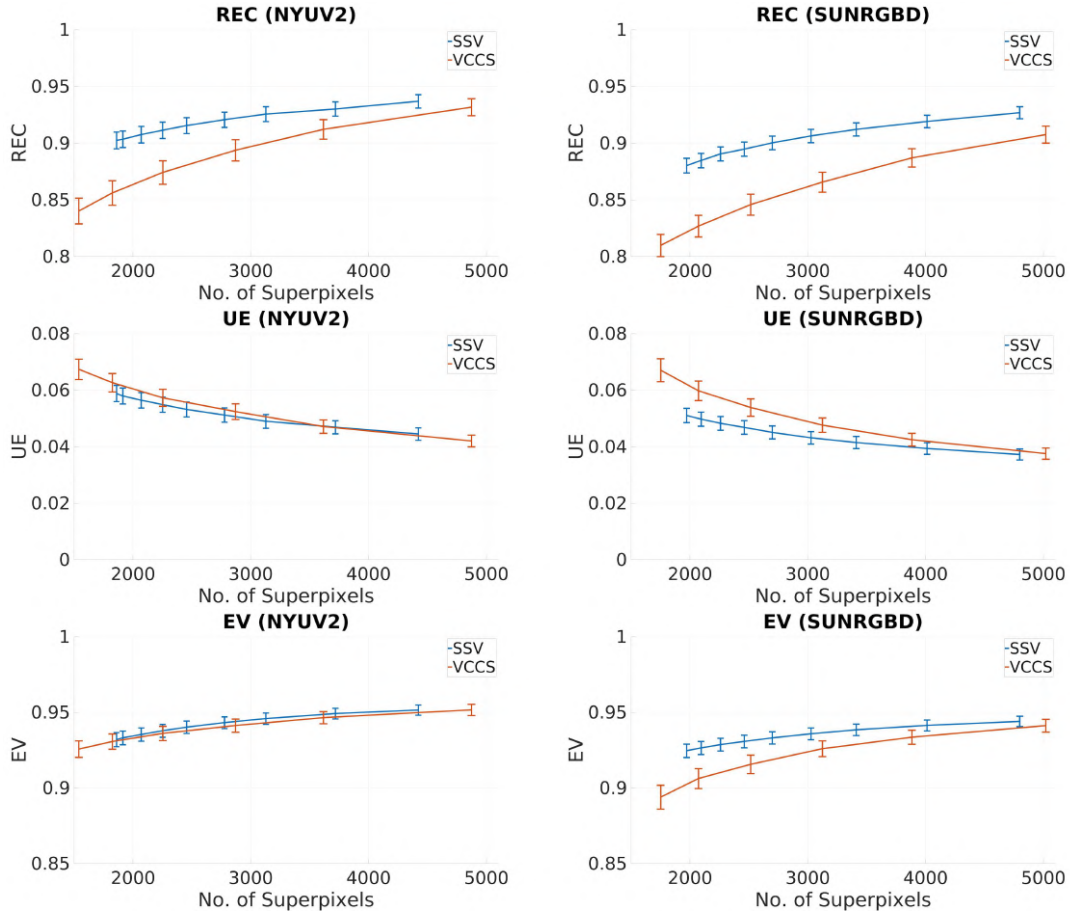


Figure 7.4: Average boundary recall (REC, higher is better), undersegmentation error (UE, lower is better) and explained variation (EV, higher is better) with 95% confidence intervals in the NYUV2 (top row) and SUNRGBD (bottom row) datasets.

are captured much better with SSV, note for example, the details of the sink (handles and tap) on the second row.

Both methods were implemented in C++, and run on a 3.5 GHz Intel i7 CPU. The average processing time of VCCS was 0.5 sec. per image. For SSV, the average processing time increases with the number of k-means clusters. With $K = 5$ clusters, the average processing time of SSV was 2.2 sec. per image. We expect that a significant speedup can be achieved by parallellizing the calls to VCCS as illustrated in Figure 7.2.

				NYUV2				SUNRGBD			
		R_{seed}		#SP	REC	UE	EV	#SP	REC	UE	EV
VCCS		0.05		10323±969	0.97±0.004	0.03±0.002	0.95±0.004	10460±720	0.95±0.006	0.03±0.002	0.94±0.006
		0.10		4873±502	0.93±0.007	0.04±0.002	0.95±0.004	5021±394	0.91±0.008	0.04±0.002	0.94±0.004
		0.15		2494±237	0.88±0.01	0.06±0.003	0.94±0.005	2773±225	0.86±0.009	0.05±0.003	0.92±0.006
		0.20		1539±143	0.84±0.01	0.07±0.004	0.93±0.006	1750±140	0.81±0.01	0.07±0.004	0.89±0.008
		0.25		1032±91	0.80±0.01	0.08±0.004	0.91±0.006	1175±89	0.77±0.01	0.09±0.005	0.87±0.01
	K	R_{min}	R_{max}	#SP	REC	UE	EV	#SP	REC	UE	EV
SSV	6	0.1	0.20	3957±341	0.94±0.006	0.04±0.002	0.95±0.003	4203±302	0.93±0.005	0.04±0.002	0.95±0.003
			0.25	3334±284	0.93±0.006	0.05±0.002	0.95±0.004	3614±257	0.92±0.006	0.04±0.002	0.94±0.004
			0.30	2949±250	0.92±0.007	0.05±0.003	0.94±0.004	3203±227	0.91±0.006	0.04±0.002	0.94±0.004
			0.35	2655±225	0.92±0.007	0.05±0.003	0.94±0.004	2899±204	0.90±0.006	0.05±0.002	0.93±0.004
			0.40	2429±206	0.91±0.008	0.06±0.003	0.94±0.004	2673±188	0.89±0.006	0.05±0.003	0.93±0.004
		0.05		4421±386	0.94±0.006	0.04±0.002	0.95±0.003	4799±333	0.93±0.005	0.04±0.002	0.94±0.004
		0.10		2949±250	0.92±0.007	0.05±0.003	0.94±0.004	3203±227	0.91±0.006	0.04±0.002	0.94±0.004
		0.15	0.3	2250±185	0.91±0.007	0.05±0.003	0.94±0.004	2460±172	0.89±0.006	0.05±0.002	0.93±0.004
		0.20		1859±149	0.90±0.007	0.06±0.003	0.93±0.005	2030±139	0.88±0.006	0.05±0.003	0.93±0.004
		0.25		1611±126	0.89±0.008	0.06±0.003	0.93±0.005	1750±118	0.87±0.007	0.05±0.003	0.92±0.005
	2			3097±296	0.89±0.009	0.05±0.003	0.94±0.004	3362±259	0.87±0.008	0.05±0.003	0.93±0.005
	3			3286±309	0.91±0.008	0.05±0.003	0.95±0.004	3558±271	0.89±0.007	0.04±0.002	0.94±0.004
	4	0.1	0.2	3481±311	0.92±0.008	0.05±0.002	0.95±0.004	3757±278	0.91±0.006	0.04±0.002	0.94±0.004
	5			3741±330	0.94±0.006	0.04±0.002	0.95±0.003	4000±293	0.92±0.006	0.04±0.002	0.94±0.004
	6			3957±341	0.94±0.006	0.04±0.002	0.95±0.003	4203±302	0.93±0.005	0.04±0.002	0.95±0.003

Table 7.1: Effect of parameters on number of superpixels (#SP), boundary recall (REC), undersegmentation error (UE), and explained variation (EV): mean ± 95% confidence interval.

7.5 Summary

In this chapter, we presented a framework for oversegmentation of point cloud data. Ensuring the high quality of RGB-D image segmentation is important in applications that require precise object boundaries, e.g., manipulation, since any segmentation errors propagate through the whole image processing pipeline and cannot be corrected in later processing stages. Motivated by this observation, we propose to apply visual saliency to guide the oversegmentation of an RGB-D image. More supervoxels are generated in highly salient regions that are likely to contain objects, while fewer supervoxels are generated in low-saliency background regions. Our approach preserves object boundaries significantly better than a current state-of-the-art supervoxel method, which generates supervoxels uniformly over the whole scene.

Apart from visual saliency, other priors such as edge information and surface normal can also guide the adaptive seeding process. Moreover, it is interesting to investigate how to integrate the over-segmentation method with the recently developed CNN based semantic segmentation method. The CNN-based methods typically use only color information and require a large amount of annotated training data. Combining both nonlearning and learning approaches can potentially lead to a framework that relies less on annotated data and also performs well.

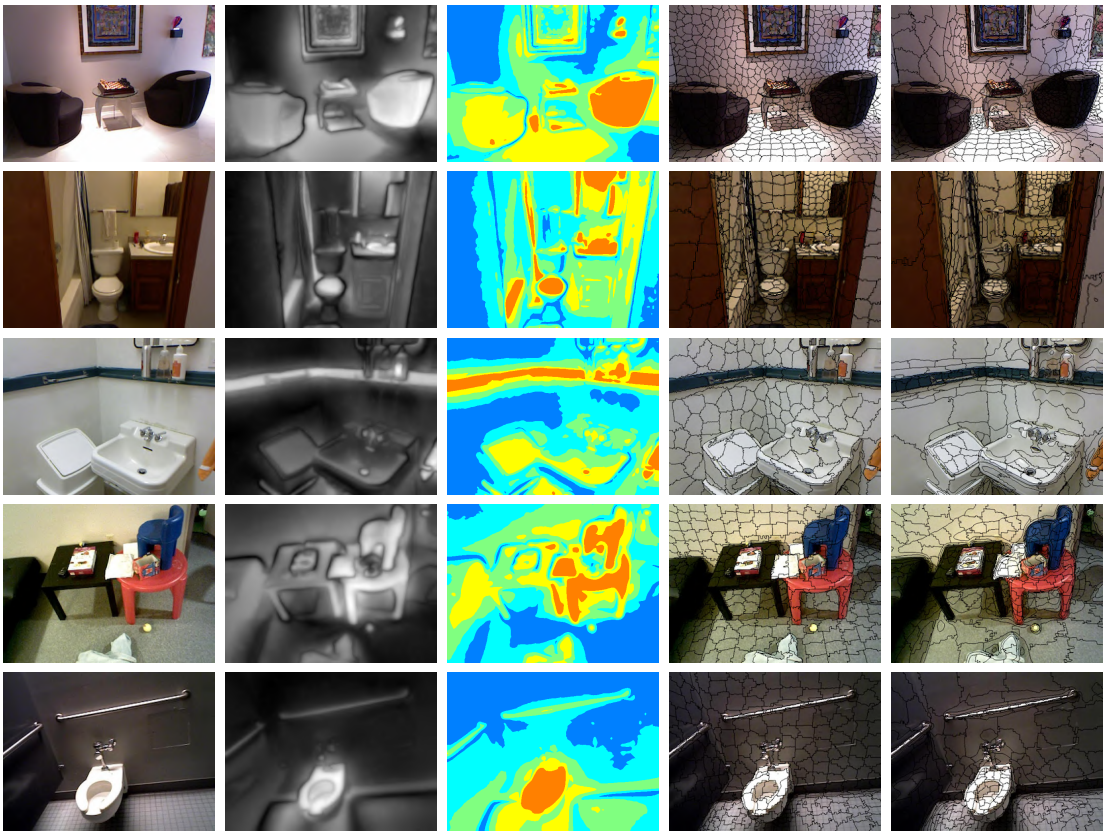


Figure 7.5: A qualitative comparison of SSV and VCCS for NYU2 (top 2 rows) and SUNRGBD (bottom 2 rows) datasets. From left to right: input image, saliency map, result of k-means clustering, supervoxels from VCCS, and our SSV supervoxels projected to the 2D image plane.

As mentioned in Section 1.2, For CloudPose and CloudAAE, the accuracy of object segmentation affects the accuracy of 6D pose estimation. A clean object segment without any foreground or background noise makes the pose estimation task much easier. For all the experiment we conducted for CloudPose and CloudAAE, we used a deep learning-based semantic segmentation for segmenting the target object. Because it is a color-based segmentation, the resulting object segments sometimes contain pixels that are physically far away from the object. Since SSV is a depth-based segmentation, it can be used to remove the noisy pixels by using the 3D pixel coordinates. However, attempting to use a non-learning-based 3D segmentation to improve a learning-based 2D segmentation is not the most practical approach. An alternative to achieve this improvement is directly using a learning-based 3D segmentation.

Chapter 8

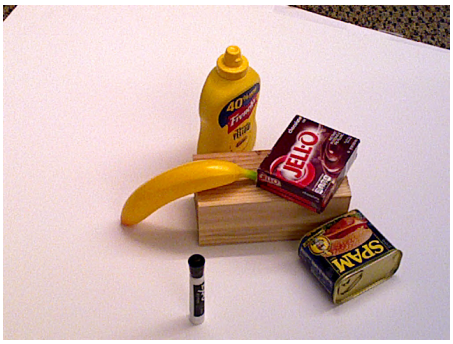
A Case Study in In-hand Pose Estimation

So far, we have introduced two methods on estimating 6D object poses from point clouds (Chapter 4, 5), one method on point cloud-based data synthesizing (Chapter 6), and one method on oversegmenting point clouds using both color and depth information (Chapter 7). As mentioned in Section 1.1.2, out of the many challenges of the 6D object pose estimation problem, one of our focuses is on decreasing the difficulty level of integrating our pose estimation system in real-world applications. In this chapter, we adapt the presented contributions for a real-world application.

We are interested in the task of in-hand object pose estimation. The "hand" in our case is referred to as the human hand. In-hand object pose estimation is to estimate the pose of a target object when it is held by a human hand [45]. Figure 8.1(a) shows an example of a mustard bottle in a human hand. This in-human-hand object pose estimation is an important prerequest for applications such as human-robot handover. Comparing to the table top scenario in the YCBV dataset 8.1(b), those two scenarios are similar in the sense that those mustard bottles are both occluded by either the human hand or foreground objects. One main difference is that the range of the target object pose in the human hand can be more diverse, hence, more challenging. Overall, the usefulness of the application, as well as the shared similarity and increased level of challenge make in-hand object pose estimation an interesting case to investigate.

In this chapter, we will use CloudSyn to create synthetic training data and train a CloudAAE network for the task of in-hand object pose estimation. Afterwards, we will collect real-world test data for testing the trained CloudAAE. The overview of this application is presented in Section 8.1. The detailed data collection and generation methods and the pose estimation system are presented in Section 8.2. Section 8.3 presents the experiment results. Section 8.4 concludes this chapter.

(a) The mustard bottle in a human hand (HOnnotate dataset [45])



(b) The mustard bottle on a table top (YCBV dataset [139])

Figure 8.1: Scenario for in-hand object pose estimation (a) and table top object pose estimation (b).

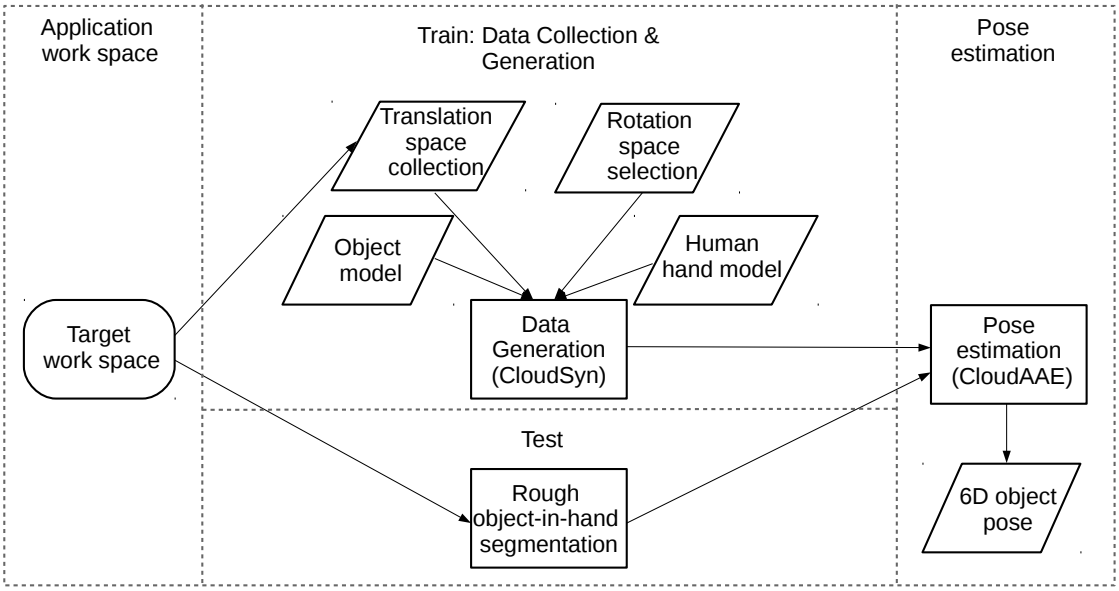


Figure 8.2: Overview of the real world experiment.

8.1 Overview

The overview of the real-world application experiment is illustrated in Figure 8.2. In general, there are three steps: the first step is to decide on a target workspace; the second is to collect the required data from the target workspace and create the synthesized training data using CloudSyn; the last step is to train a pose estimation system CloudAAE with the synthesized data. A target workspace means a particular real-world setup that may contain, e.g., a robot and a table, and a camera, either mounted on the robotic hand or on the side, viewing the whole workspace.

In the data collection and generation block, we prepare the required component for data generation with CloudSyn. As it is illustrated in Figure 8.2, the only data

Figure 8.3: Setup of the real world experiment.

to be collected from the target workspace is the translation space. Details regarding the translation space collection are presented in Section 8.2.1. To ease the process of data collection and ensure good coverage in the rotation space, the rotation space is to be selected according to specific heuristics. Details for determining the rotation space are presented in Section 8.2.2. The object models and human hand models are directly obtained from the corresponding data sets. To prepare the object and hand models for data generation, the relative positions should be determined. This process is described in Section 8.2.3.

After deciding the relative poses between object and human hand models, the poses from prepared rotation and translation space are used for transforming the object and hand models. The process from here on is the same as the data synthesis pipeline CloudSyn presented in Chapter 6 (Section 8.2.4). Finally, the generated data is used to train the pose estimation system CloudAAE (Section 8.3.1). During the testing phase, the target human hand and object are roughly segmented and fed into a trained network for pose estimation. This is described in Section 8.3.2.

8.2 Data Preparation

In this section, we describe the details regarding data preparation during the training phase. The experiment setup is shown in Figure 8.3. We use a Kinect V2 camera to view a workspace containing a table and a robotic arm. This is the target workspace used for data collection. Section 8.2.1 and 8.2.2 describe how the translation and rotation space are obtained. Section 8.2.3 describes how to decide the relative pose between the human hand and object models. Section 8.2.4 describes how those components are used in the data synthesis pipeline for training a CloudAAE network.

8.2.1 Translation Space Collection

We use AprilTag [95] to collect the range of the translation space. AprilTag is a visual fiducial marker that uses a 2D bar code style and allows full 6-DoF localization of the target tag from a single image [95]. Figure 8.4(a) shows an example of the AprilTag attached to a box, and we use this for pose collection. Since we are only interested in collecting the 3D translation poses from the workspace, we opt for this simple approach. Another option is to attach one or several AprilTags on the target object for pose collection. However, since many target object such as the power drill has mostly curvy surfaces, the attached AprilTags should be placed on hard plastic pieces or cardboard to avoid deformation. This is significantly more complicated than printing the tag on paper and attach it to a box. Furthermore, this would require attaching the tags to each target object which further increases data preparation complexity.

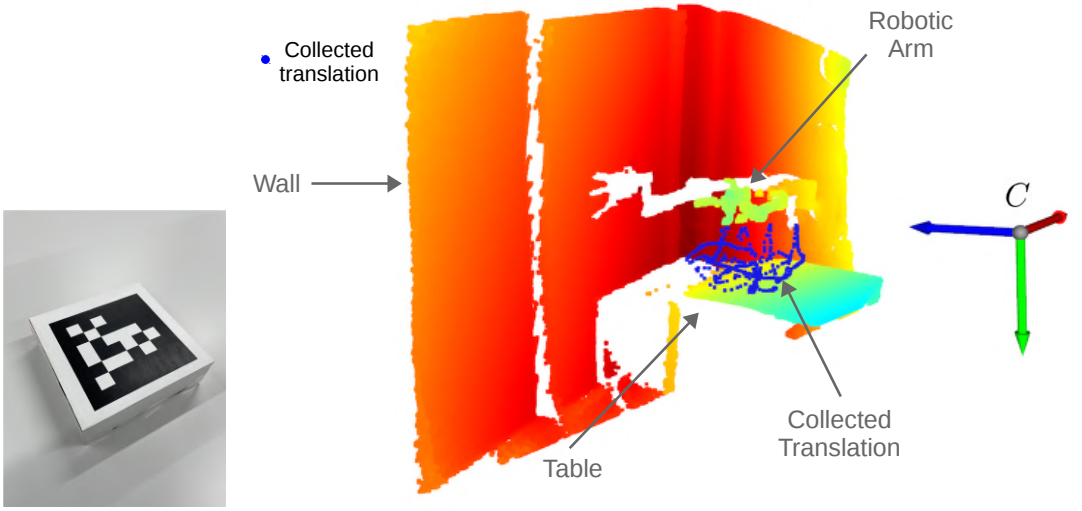
Figure 8.4(b) shows the collected translation and the work space presented in 3D point clouds. We show the AprilTag at various locations in the workspace to the camera C and plot the detected translation in blue dots. Furthermore, we denote the robotic arm, wall, and table in the workspace. The goal of the translation collection is to roughly cover the application space and the boundaries that define the space.

After collecting the translation range, we calculate the kernel density estimate of the collected poses. Afterwards, we draw a number of samples from the density estimation. This process is illustrated in Figure 8.5(a), where the collected translation is denoted in blue and the sampled translation is denoted in magenta. Figure 8.5(a) shows the collected and sampled translations in the work space from different views. It can be seen that the sampled pose provides a good coverage of the desired workspace.

8.2.2 Rotation Space Selection

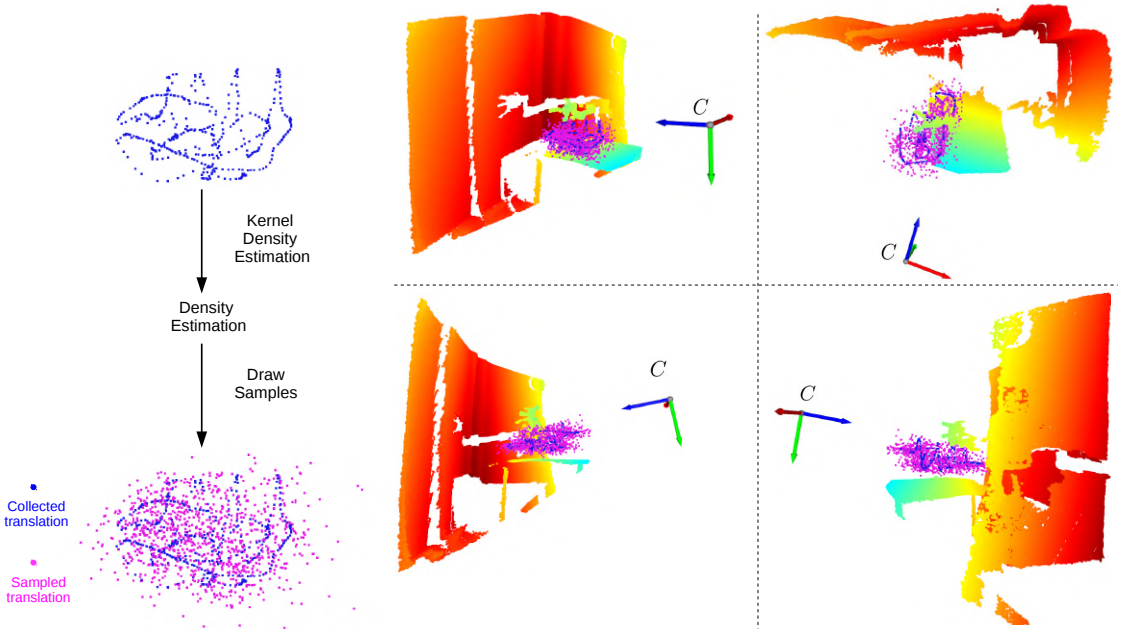
In this experiment, we decided that the rotation space is not related to the data collection process and should be selected based on some heuristics. The main reason is that compared to translation collection, it is challenging to have good coverage of the target rotation space during data collection. The target translation space is well-defined by the workspace setup, e.g., the translation space should be above the table rather than under the table. In comparison, the target rotation space is set by the user, and theoretically, it can contain any arbitrary 3D rotations. Another reason is that due to self-occlusion, the AprilTag can not be fully viewed by the scene camera in certain 3D rotations; hence the corresponding poses cannot be detected.

Ideally, the selection procedure should be as easy as possible so that the cost of generating synthetic data is low. The easiest choice would be random samples from the 3D rotation space. As it is described in Section 2.2, a 3D rotation can be represented by a unit vector $\hat{\mathbf{e}} \in \mathbb{R}^3$ and a scalar $\theta \in \mathbb{R}$. Therefore, we randomly sample a unit vector $\hat{\mathbf{e}}$ and a scalar $\theta \in [0, \pi)$. The reason that π is excluded is because



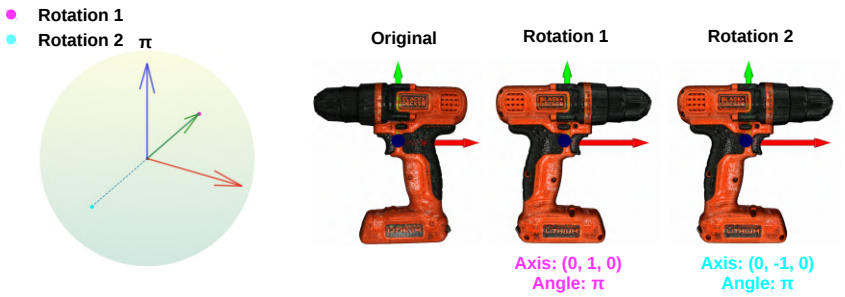
(a) An AprilTag (b) Collected translation

Figure 8.4: Translation collection.(a) The example of an AprilTag. (b) Illustration of collected translation.

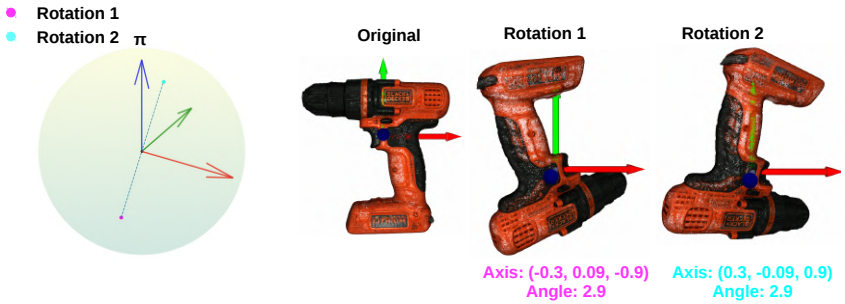


(a) KDE of translation (b) Sampled translation

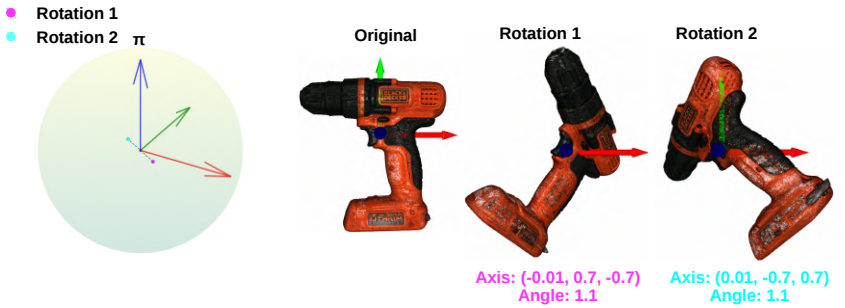
Figure 8.5: Translation sampling.(a) Kernel density estimation. (b) Illustration of sampled translation.



(a)



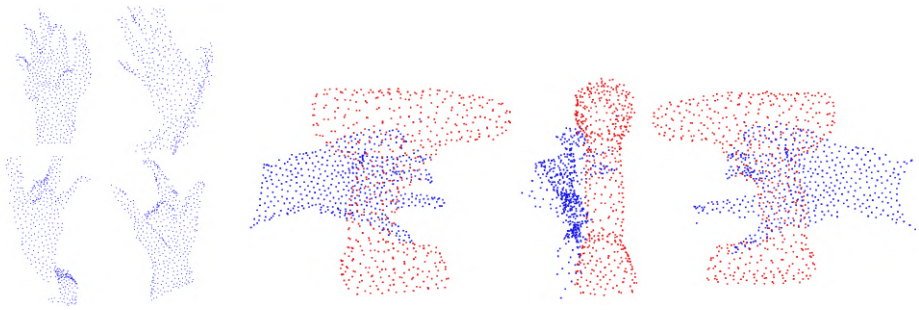
(b)



(c)

Figure 8.6: Example of 3D rotations. (a) If two rotation directions are antipodal and the rotation angles are both π, they result in the same rotation. (b) If two rotation directions are antipodal but the rotation angles are less than π, they result in different rotations. (c) If two rotation directions are not antipodal and the rotation angles are less than π, they result in different rotations.

the resulting 3D rotation corresponds to two different axis angle representations $\theta\hat{\mathbf{e}}$ and -$\theta\hat{\mathbf{e}}$. This may potentially lead to creating two identical object segments with two different 3D pose annotations. This is not desired since the pose network is learning a one-to-one mapping from the input to the output. Figure 8.6(a) shows a corresponding example. The two rotations are denoted with the colored dot in a sphere with a radius of π. They have the same amount of rotation (π) but different unit vectors. However, as shown, those two axis-angle representations result in the same 3D rotation for the target object. Figure 8.6(b) and 8.6(c) show two other examples where the rotation angle is less than π. In this case, two opposite rotation axes result in two different 3D rotations and can be used for network training.



(a) BigHand (b) Object and hand model

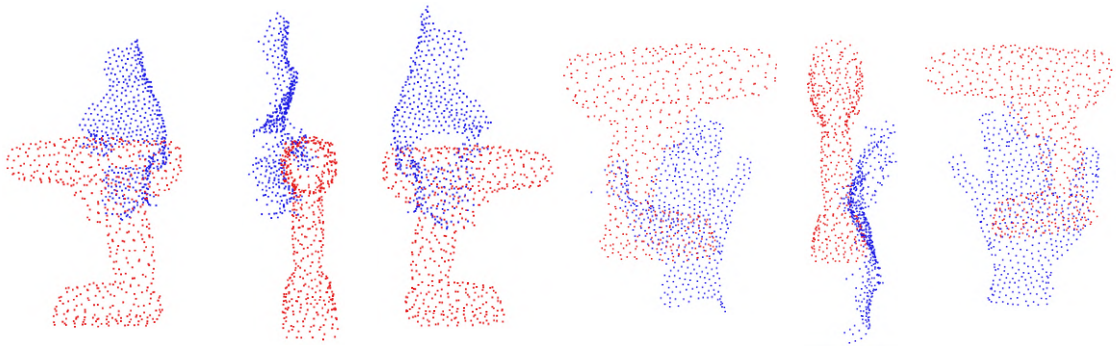
Figure 8.7: Examples of object and hand models. (a) Examples of hand model from the BigHand data set. (b) One example of object and hand model in three different views.

8.2.3 Object and Human Hand Model

To create a synthesized data set for in-hand object pose estimation, we need the object model and the human hand model. Since we use the object from the YCB video dataset, the object model can be obtained from the same data set. The human hand model is obtained from the "BigHand" data set [146]. Figure 8.7(a) shows four examples of the human hand model in the BigHand data set. In general, those hand models are human hands in various poses and captured by a depth camera.

The relative position between the hand model and the object model must be determined for synthesizing data. This requires some handcrafting according to the object model dimensions. Figure 8.7(b) shows an example that mimics the scenario of a human hand holding a power drill by the handle. We show this example in three different views. Although the poses of the finger positions are not very realistic, it roughly simulates the desired object-in-hand scenario. In general, it is a decent synthetic example generated at very low cost. To add more variance to the relative pose between object and hand models, we also generate other instances where the human hand is positioned at a different part of the drill, as shown in Figure 8.8. Figure 8.8(a) shows an example of the human hand "holding" the head of the drill, and Figure 8.8(b) shows an example of the human hand "holding" the bottom part of the drill.

So far, we have only shown those cases where both the hand shape and the relative poses produce examples that rather highly resemble the real scenario. However, this is not always the case. There are also unrealistic samples, as shown in Figure 8.9. The main reasons for those unrealistic examples are the undesired pose of the hand model as well as the relative pose between the hand and the object. In general, we are able to generate the hand and object data at a very low cost, and we expect the large amount of data being able to compensate for the impact of the low-quality synthetic data.



(a) (b)

Figure 8.8: Two examples of object and hand models in three different views. (a) Human hand "holding" the head of the drill. (b) Human hand "hodling" the bottom part of the drill

Figure 8.9: Some examples of unrealistic positions.

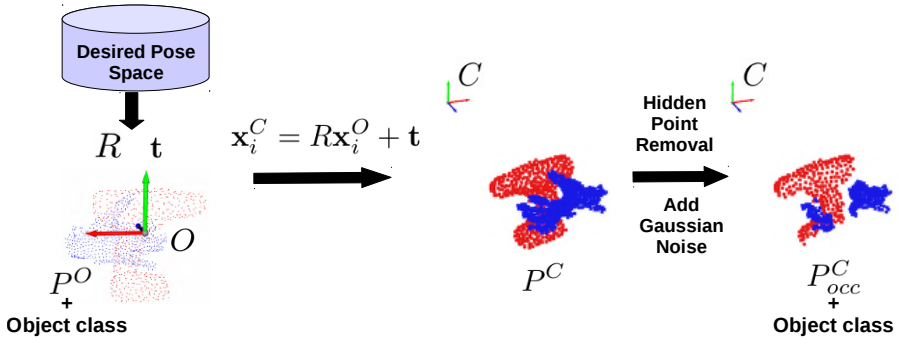


Figure 8.10: Data generation pipeline adapted for in-hand object pose estimation.

8.2.4 Data Generation

So far, we have the 3D translation space obtained by data collection, the 3D rotation space from selection, and the relative poses between the object and hand model. Now we can draw a 3D translation $\mathbf{t}$ and 3D rotation R as depicted in Figure 6.5. We denote the point cloud containing the hand and object model as P^O. Following the data synthesizing pipeline, the next step is to apply the transformation $\mathbf{x}_i^C = R\mathbf{x}_i^O + \mathbf{t}$ to each point $\mathbf{x}_i^O \in P^O$ to obtain $P^C = \{\mathbf{x}_i^C \in \mathbb{R}^3 \mid i = 1, 2, \ldots, n\}$. Then, hidden point removal and Gaussian noise are applied

to P^C to obtain the final object segment P^C_{occ} for training. Figure 8.10 illustrates this pipeline. The final segment P^C_{occ} is normalized and fed into a network with the structure of CloudAAE (Chapter 5). The color code illustrated in Figure 8.10 is only for visualization. All points are treated as the object point during training.

The data generation process takes around 19 minutes for generating $10,000$ training samples. Each training sample contains the 3D coordinates of 256 points, the 3D translation, the 3D rotation, and the object class information. The $10,000$ samples in total take approximately 30 MB of storage. We generate $30,000$ training samples and train the network for 100 epochs. The training takes 3 hours on an Nvidia Titan X GPU.

8.3 Experiment

In this section, we present how a trained network is used during the testing phase and the results of the experiments. We present the training results in Section 8.3.1. We describe how to obtain test data that is similar to our synthetic training data in Section 8.3.2. The test results are shown and discussed in Section 8.3.3.

8.3.1 Network Training

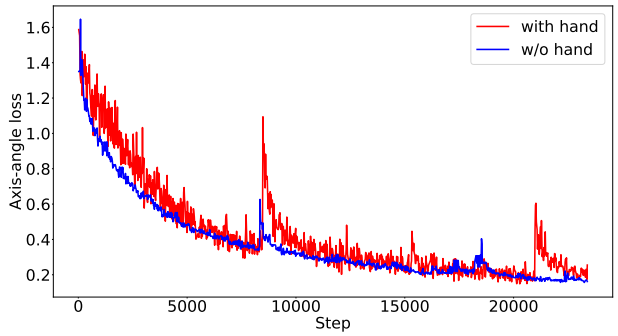
We train a network for the object power drill from the YCB video dataset. Figure 8.11 shows the training losses in red. The general trend of losses decreases during training, and all losses converged at the end of the training.

Figure 8.12 shows some visualization of the training results. We transform the input training segment (red) with the predicted 6D poses and overlay it with the object model (green) (Figure 8.12(a)). The corresponding object reconstructions (blue) are presented in Figure 8.12(b). Together with the training loss and visualization, it can be seen that the network learned how to regress to an accurate 6D pose after training.

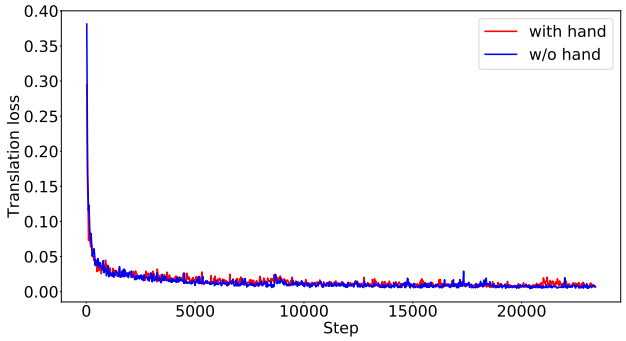
8.3.2 Test Data

During testing, a human is holding the target object in the workspace, and a camera is used to capture the scene. To obtain the segmentation of the human hand and object, we leverage the recently developed semantic segmentation network [150] trained on the EgoHands dataset [5]. By detecting the hand region on the RGB image, we are able to crop the image region containing both the hand and object.

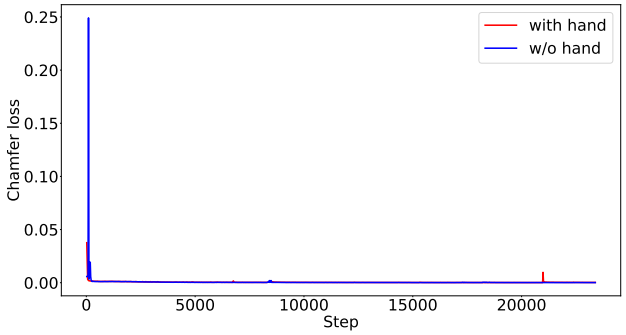
Figure 8.13 shows some examples of segmented hand and object. Due to the inaccurate calibration between the color and depth images, the presented examples show a mismatch between the color and depth information. However, this is not a problem for testing our system since we use only the depth information. We use the segmented hand and object point clouds as the input to our trained pose estimation system.



(a) Rotation loss



(b) Translation loss

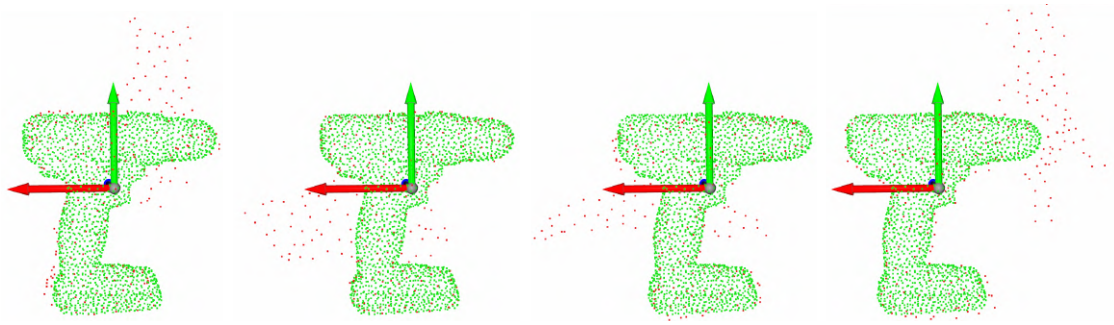


(c) Reconstruction loss

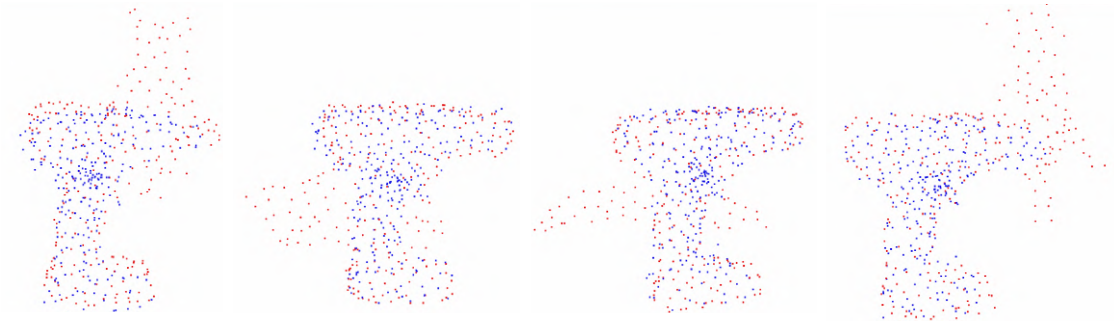
Figure 8.11: Training losses.

8.3.3 In-hand Object Pose Estimation

We show some preliminary testing results for this application. Since we do not have annotations of ground truth 6D objects for the test data, we evaluate our results by visual inspection. We transform the input segments with predicted 6D poses and inspect how they are overlaid with the object models. Figure 8.14 shows examples of the results. The colored point cloud segments are the input, and the object model is shown in green points. Those results are after ICP pose refinement.



(a) Overlaying the transformed input training segment (red) with the object model (green).



(b) Input segment (red) and object reconstruction (blue)

Figure 8.12: Visualization of training results.

Figure 8.13: Examples of segmented hand and object during testing.

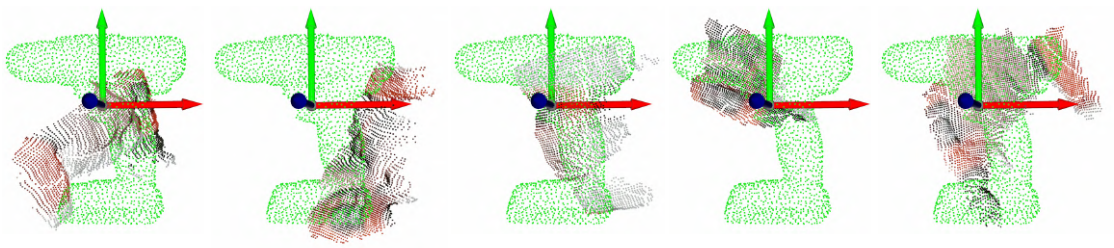
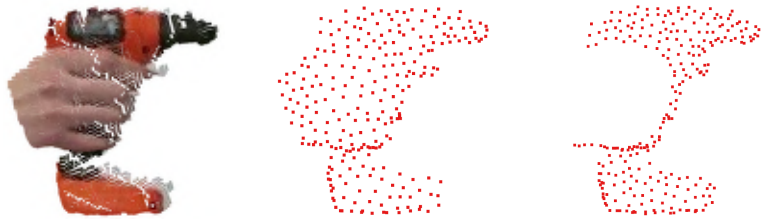


Figure 8.14: Examples of the testing result for including hand segments.



(a) Original segment with color. (b) Downsampled segment with hand. (c) Downsampled segment without hand.

Figure 8.15: Examples of the test segment with and without the human hand.

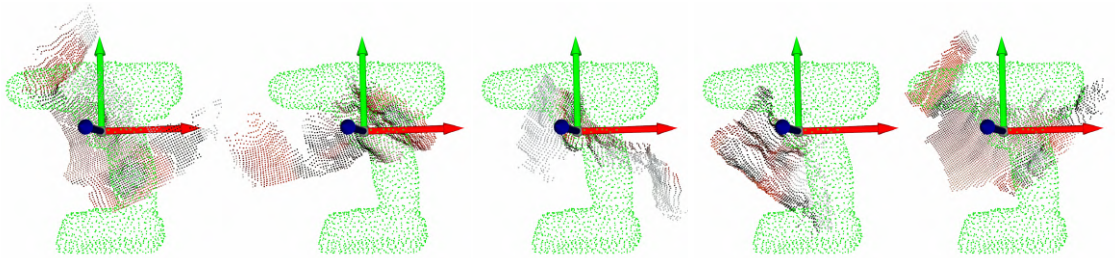
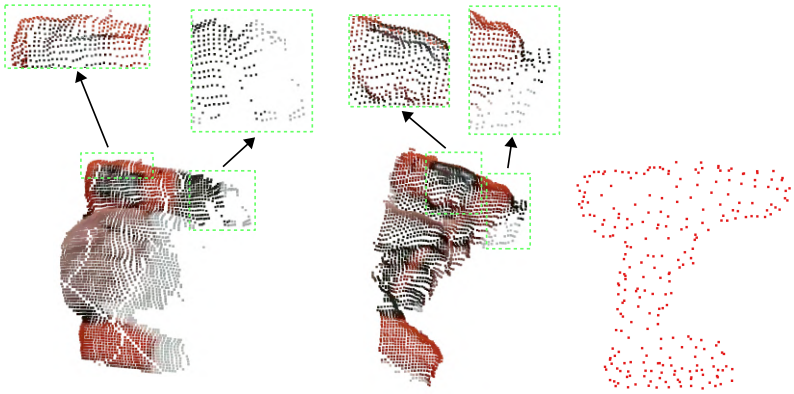


Figure 8.16: Examples of the testing result for excluding hand segments.



(a) Test data. (b) Train data.

Figure 8.17: Illustration of wavy noise in test data.

It can be seen that the pose estimation results are far from being accurate. The translation estimates are marginally acceptable, but the rotation estimates are not accurate.

Since the test result for in-hand object pose estimation is not good, we attempt to simplify the task by excluding the hand segment in both training and testing phase. Figure 8.15 show an example of an original test segment, and its corresponding downsampled version with and without the hand segment. The training losses are shown in blue in Figure 8.11. And the test results are shown in Figure 8.16.

It can be seen that the test results similar and the rotation estimates are still not acceptable.

One possible reason for this inaccurate estimation is the visual reality gap between the synthetic data and the real testing data. As we observed, the test data has some noise in the depth data, and the surfaces of the object regions with black color present a wavy structure. Figure 8.17 shows an illustration of this observation. Since the object model, we used to generate training data has a very smooth surface and only a small amount of Gaussian noise is added per point, the training segments still represent smooth surfaces. Figure 8.17(b) shows a training example. Although CloudAAE is expected to be robust against noisy data, this robustness depends on the amount of per point noise augmentation for the data synthesis process. Increasing the amount of Gaussian noise when generating training data could potentially improve the performance. Another idea for bridging this visual reality gap is to create a 3D object model of the object with the Kinect camera and use this model for data synthesis. In this way, the sensor noise is contained in both the training and testing data. Thus the visual reality gap becomes much smaller.

8.4 Summary

In this chapter we adapted our data synthesis pipeline CloudSyn and augmented autoencoder based 6D pose estimator CloudAAE for an in-hand object pose estimation application. We use the object models from the YCB video dataset and the human hand models from the BigHand dataset for creating synthetic data. The translation space is determined by recording the workspace boundaries using an AprilTag. The rotation space is selected from the full 3D rotation space. We train a CloudAAE network with synthetic data and test it with real test data collected from the same workspace. As preliminary results we inspect the pose estimation accuracy with a visual inspection.

The data synthesizing and training takes around 4 hours in total. This shows that our system is easy to adapt and lightweight in training. This is desired in real-life experiments as it iterates fast and allows rapid development. However, the preliminary results are not satisfying, and there is still room for improvement.

As mentioned above, the visual reality gap still seems to be the main reason. This can potentially be mitigated by either using a depth camera with better depth quality or making the synthetic human hand more realistic. Another open question is the selection of the rotation space. In this experiment, we only tested on the object power drill, which is not a rotational symmetric object. In this case, using the full 3D rotation space does not cause ambiguity during the training. However, there are objects with rotational symmetry structures, thus the rotation space should be selected accordingly to avoid sending mixed error signals during the training process. We also would like to point out that, rather than visual inspection, the system can also be evaluated on a downstream robot-human hand-over task.

Chapter 9

Conclusion

In this thesis, we have presented several contributions mainly to the field of 6D object pose estimation and one contribution to the field of oversegmentation. In this chapter, we will summarize the contributions in Section 9.1. Furthermore, we will discuss the strengths and limitations in Section 9.2. We will touch on the potential future work directions in Section 9.3.

9.1 Summary

At the beginning of this thesis research, we noticed that existing methods in the field of pose estimation mainly rely on color for estimating object poses. In comparison, the depth information was often ignored or regarded as auxiliary information. This motivated us to investigate the depth information and see how it can contribute to 6D object pose estimation. We want to affirm that it is most beneficial to use data from multiple modalities in a system because each modality has its own advantages. The caveat is that each modality should be used appropriately to exploit its strengths and avoid its weaknesses. Hence, this was another motivation for us to investigate depth information in-depth.

The first contribution is CloudPose, which is a point cloud-based system for 6D object pose regression. Since it was the first work on 6D object pose regression from depth information, we mainly revisited four fundamental aspects for designing the system. We made four main findings in this work. Firstly, we chose point cloud as the appropriate depth representation because it contains the full 3D information. Secondly, since the 3D rotation and 3D translation were inherently two different metrics, it was reasonable to regress them with separate networks. Thirdly, we argued that because axis-angle was a constraint-free rotation representation, it was the most suitable regression target in a supervised learning framework. Lastly, a proper distance measure for rotation loss was the Geodesic distance. Experimental results showed that our network was structurally simple but performed well and achieved the state-of-the-art performance on the largest dataset.

The second contribution is CloudAAE. During the evaluation of CloudPose, we noticed that it was susceptible to noisy depth data. One possible reason was the

information sparsity in the input data. Since the input data had approximately 6000 dimensions while the output had only 6 dimensions, the pose data was very sparsely encoded in the input data. This made it difficult for CloudPose to extract the essential information about the object pose. Hence, we decided to use an autoencoder based structure for dimension reduction and information distillation. We proposed CloudAAE, for which we adapted an existing augmented autoencoder idea for a point cloud-based system. By inputting a noisy and occluded point cloud segment and reconstructing a noise and occlusion free point cloud segment at the same 6D pose, we learned a low-dimensional latent code that explicitly encoded the pose information. This latent code was used as input to two separate networks for rotation and translation regression, respectively. Experimental results showed that this structure was more resistant to noisy input data and can produce better pose estimation results.

The third contribuction is CloudSyn. Since the two systems mentioned above relied on supervised learning, they needed a large amount of annotated data for training. Due to the high cost of annotating 6D object pose on real-world data, it was preferred to generate synthetic training data. Existing data generation methods were mainly color-based, and it was relatively expensive to bridge the visual reality gap. Since we had shown that we could regress the accurate object pose from depth information, we would also like to have a depth-based data generation pipeline. A major advantage of this was the low cost of data generation, thanks to the small visual reality gap for depth information. To this end, we presented CloudSyn, which was a point cloud-based pipeline for synthesizing training data. The required inputs were a texture-less 3D object model and a 6D pose drawn from the desired pose spaces. After transforming the object model to the desired 6D pose, spherical occluders were added between the object model and camera center for simulating external occlusions. We used hidden point removal and added per point noise to generate the final synthesized point cloud segment. By combining CloudSyn with CloudAAE, we could fully utilize the "augmented" aspect of CloudAAE since CloudSyn provided data augmentation that we would like CloudAAE to learn and to ignore. CloudSyn was cheap and fast, and its synthetic data could be used to train CloudAAE to achieve the state-of-the-art performance on the LineMOD dataset.

The fourth contribution is SSV, which is short for saliency-guided adaptive seeding for supervoxel segmentation. So far, we have assumed that the target object segmentation is available to be used as the input to our pose estimation systems. Practically, the segments can easily be obtained using off-the-shelf CNN-based semantic segmentation methods. During evaluating CloudPose and CloudAAE, we noticed that the quality of the segment had an impact on the system performance. Specifically, a clean segment without incorrectly segmented background fragments would lead to a more accurate pose estimate. Since the CNN-based semantic segmentation methods are usually using only color information, they sometimes mistakenly include background fragments. This can potentially be trimmed with the help of depth-based segmentation. To this end, we present SSV, a point cloud over-segmentation framework with saliency-guided adaptive seeding. We used vi-

sual saliency from color information to guide the initial seeding process and put denser seeds in more salient regions while coarser seeds are used in less salient regions. This resulted in smaller supervoxels in more salient regions and bigger supervoxels in less salient regions. Experimental results showed that this could better preserve the object boundaries while allowing more accurate object segmentation.

So far, we evaluated our system with datasets containing tabletop scenarios. To examine whether our pose estimation and data generation systems could be easily adapted for a slightly different but very relevant task, we presented a case study for a real-world in-hand object pose estimation application. We combined CloudSyn and CloudAAE for this task. We adapted CloudSyn to generate in-hand object pose estimation training data. We trained CloudAAE with synthetic data and tested it with real-world data. Although the preliminary results were not satisfying, the implementation process showed that our system was easy to adapt and lightweight in training. This is desired in real-life experiments as it iterates fast and allows rapid development.

Recall from Section 1.1.2, where we stated that we would like to focus on developing a lightweight system with a good ability to scale up to many objects. The multi-class system nature of CloudPose and CloudAAE makes them suitable for handling a number of objects with the same network while maintaining a good pose estimation accuracy. CloudSyn also helps reducing the difficulty level of applying a pose estimation system in real-world scenarios. While there is still plenty of room for improvement in performance, we believe this thesis help to take a concrete step in the right direction.

9.2 Strengths and Limitations

While maintaining good performance accuracy on public benchmarks, one of our systems' main strengths is that they are generally lightweight. For example, to train a network with CloudPose structure for 21 target objects takes around 22 hours on an Nvidia Titan X GPU. This is a very short training time on a single GPU for a deep learning-based system. This enables other researchers who have limited GPU resources to test and retrain the system easily and enables faster research iterations. The main reason for our systems being lightweight is that they generally have very simple structures. For example, the main component of CloudPose is two separate networks that directly regress 3D rotation and 3D translation from a point cloud segment. This leads to the second strength that it is relatively easy to investigate the system's explainability by directly looking at the points that contribute to the pose estimation. This is done by visualizing the active points that the networks used for estimating 6D poses, as presented in Section 4.5.5. This is an important path towards safe and robust systems. The third advantage is that we can generate a large amount of training data at a very low time and hardware storage cost with CloudSyn. This low cost also enables fast research iterations when applying our methods to estimate the pose of, for example, a set of new objects. The fourth advantage is that since our pose inference stage only uses depth data,

our methods are more robust to illumination changes than methods that rely on RGB information.

The simplicity of the system not only accompanies with strengths but also comes with some limitations. The main limitation is that our systems do not use color information. Although color information usually comes with challenges such as illumination vulnerability and a large visual-reality gap, it undoubtedly has many advantages. With its compact matrix format, it provides rich texture information that depth information fails to capture. Moreover, it has been shown in many visual recognition fields that deep CNNs can extract useful features at different abstract levels from color information. It can also be seen from the experimental results that the state-of-the-art methods use both color and depth information to achieve good performance. In comparison, using only depth information seems to hit a glass ceiling on the performance scores. Another limitation is that a depth sensor is required to capture the depth data. Although depth sensors are very popular nowadays, they are still less prevalent than RGB cameras.

9.3 Future Work

This work could be extended in several aspects, and we will mainly discuss two aspects in this section. The first is from the data modality aspect. Till this point, we have much experience of deep learning on depth data, and it is time to merge the power of depth with other data modalities such as color (Section 9.3.1). The second aspect is from the problem of 6D pose estimation. One of the challenges mentioned in Section 1.1.2 is the pose ambiguity of rotational symmetric objects. Since our method employs only depth information, it can potentially be used to investigate the pose ambiguity with purely geometric information. Detailed descriptions are in Section 9.3.2.

9.3.1 Multimodal Data

One obvious and essential future extension is to add color as another modality. The key to using multimodal data is how to fuse them. So far, there are a handful of RGB-D based methods [131, 49] that have a data fusion procedure. The fusion strategy they use is simple concatenation, and this already gives very good testing results on the public datasets. On the other hand, more complicated depth and color fusion strategies were proposed in other fields such as RGB-D salient object detection [81] and cross-modality person re-identification [85]. It would be interesting to integrate those more sophisticated data fusion methods into 6D object pose estimation.

However, one problem with this is that the datasets for 6D pose estimation are close to being "solved". The recent state-of-the-art method PVN3D [49] achieves 99.4% accuracy on the LM dataset and 96.1% on the YCBV dataset. Since a simple strategy can already achieve such good performance on those datasets, they are not suitable to measure improvement using a more complicated fusion strategy. One

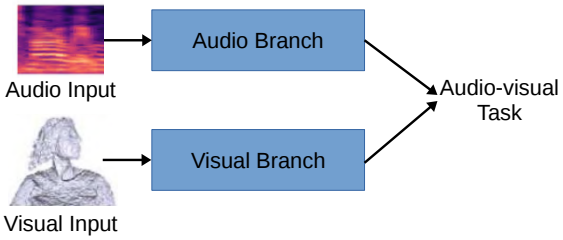


Figure 9.1: Illustration of an audio-visual task. Figure adapted from [11]. The visual input is an example from the BIWI dataset [29].

solution to this is to have a more challenging dataset for this purpose. Another potentially more fruitful approach is to use our knowledge on depth-based deep learning in other relevant robot vision fields. One interesting field is 3D object detection [153], which is also an important research topic in robot vision. The main difference between 3D object detection and 6D object pose estimation is that the former deals with unknown objects while the latter deals with known objects. The current state-of-the-art performance on benchmark dataset KITTI [40] is around 70% in the hard category. This shows that there is still much room for improvement and the benchmark still has the capacity to capture those improvements. In general, we can keep gaining more experience regarding depth-based deep learning and have suitable datasets to measure the performance gain of those new systems.

Apart from purely vision-related tasks, this thesis work can also be deployed in audio-visual tasks. For example, knowing the speaker's head pose is helpful for speech recognition [7] and speech acquisition [113]. Hence, we can adapt our object pose estimation system into a human head pose estimation system and combine it with an audio branch for tasks such as speech recognition. Figure 9.1 shows an illustration. The audio branch is used for processing the audio input, while the visual branch is used for processing the visual input. The output from two branches can be used for various audio-visual tasks. Similarly, our object pose estimation system can also be combined with other data modalities such as tactile data for the robotic grasping task [18]. In general, the work in this thesis can be extended in many ways in the scope of using multimodal data.

9.3.2 Pose Ambiguity

The pose ambiguity problem in a learning system arises from the pose labels that are annotated based on the pre-defined object coordinates. However, for rotational symmetric objects, different object coordinate poses can correspond to the same object appearances. For a deep learning system, the same object appearances mean the same network input. When training a network with the same inputs corresponding to two different pose labels, the network receives mixed training signals and may have trouble with convergence.

The existing works on pose ambiguity are mainly based on color information [21, 100]. This renders the pose ambiguity problem ill-posed because object

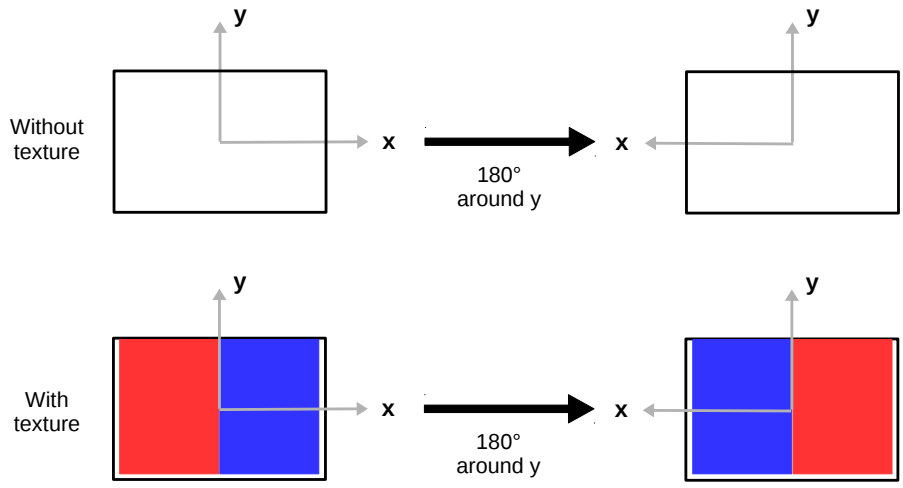


Figure 9.2: Illustration of object rotation with and without texture.

symmetry is a property of geometric shape rather than texture. Figure 9.2 shows an illustration. For the texture-less rectangular object, when rotating 180° around its y-axis, the resulting object has the same appearance because of its symmetric shape. For the textured rectangle, shown with the red and blue texture, the same rotation will result in the object with a different appearance in a different pose. Theoretically, for color-based approaches, this would not cause ambiguous training signals. Hence, the problem of pose ambiguity does not fully present itself in color-based approaches. In contrast, this problem fully occurs in depth-based approaches, and it is more suitable to be investigated with depth-based approaches.

In general, if the above-mentioned extensions are integrated into either CloudPose or CloudAAE, it is very likely that a robot can bring you the morning coffee smoothly. Moreover, if the problem of 6D object pose estimation is solved, and hopefully the same for robotic grasping and dexterous manipulation, then a robot can also bring you a self-made breakfast along with the coffee. Let's fast forward to the year 2050, while you are enjoying the morning coffee and breakfast served by your newly purchased household robot, you might be thinking: "What a time to be alive!"

Appendix A

Basics of Point Cloud and 3D Rotation

The aim of this chapter is to provide the reader with some basic information on point cloud and 3D rotations. Regarding the point cloud, we explain how a point cloud can be obtained from intrinsic camera parameters and the corresponding depth image. Regarding the 3D rotations, we describe the details regarding the 3D rotation group and the conversion among different rotation representations.

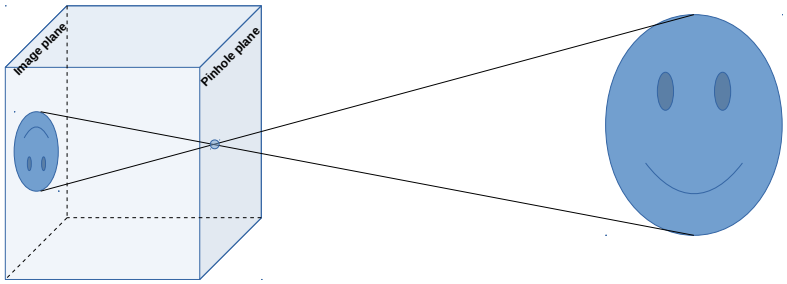
Section A.1 introduces the relationship between depth images and point clouds. It starts with introducing the pinhole camera model (A.1.1), then moving on to explain depth sensing technology and depth images (A.1.2). Afterwards, it describes how to obtain point clouds from depth images using the pinhole camera model (A.1.3). Section A.2 introduces the 3D rotation group, as well as the conversions among commonly used rotation representations.

A.1 Depth Map and Point Cloud

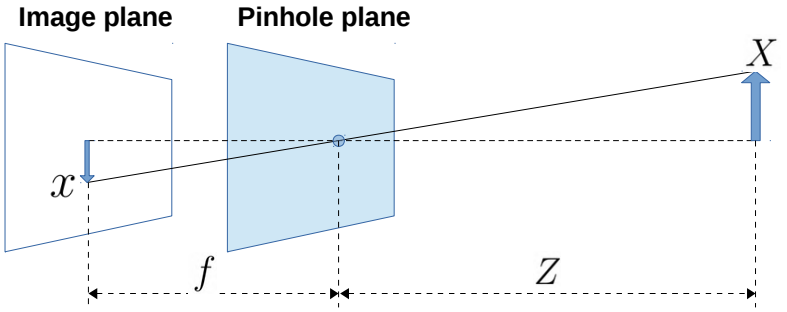
A 2D color image is created by projecting the 3D world to a 2D plane using a camera. During this projection, the scene appearance is well preserved by the color information. However, this projection causes the loss of physical distance data, which contains information such as the physical size and position of an object in the scene. Without those information, a robot can not interact with its surroundings properly. Luckily, a depth map contains the distance information. If knowing the details about the camera which is used for imaging, the full 3D world information can be recovered from the depth map. Before delving into the details about depth images, we first introduce the pinhole camera model which provides more insights of the 2D-3D projection.

A.1.1 Pinhole Camera Model

The pinhole camera model is the simplest model of a camera [16]. Figure A.1(a) shows an illustration of a pinhole camera. It can be depicted as simple as a box, of



(a) A pinhole camera.



(b) The pinhole camera model.

Figure A.1: An illustration of a pinhole camera.

which the front plane is the pinhole plane and the back plane is the image plane. A pinhole is made in the center of the pinhole plane, and this pinhole is functioned as the tiny aperture. From each point of the real world object on the right, there is exactly one ray of light that travels from the physical point through the pinhole and reaches the image plane. This is the process of project a 3D point onto a 2D image plane.

From the pinhole camera, its model is derived (Figure A.1(b)) [16]. The distance between the pinhole plane and the image plane is called the focal length, and it is denoted as f. Assuming X is the object length, x is the object's image on the image plane, the relationship between X and x is

$$-x = f\,\frac{X}{Z} \tag{A.1}$$

in which Z is the distance between the pinhole plane and the object.

For a more convenient and equivalent mathematical representation, the pinhole camera model is rearranged by swapping the pinhole and the image plane, as illustrated in Figure A.2. With the rearrange, the project object becomes right-side up. The pinhole is now behind the image plane and it is referred as the **center of projection**. The line from center of projection and it is perpendicular to the image plane is the **optical axis**. The intersecting point between the optical axis and the image plane is the **principle point**. The 3D world coordinate is denoted with $(\hat{i}, \hat{j}, \hat{k})$ and the image coordinate is denoted with $(\hat{u}, \hat{v})$.

Ideally, the principle point should be the exact center of the image plane, in practice this is usually not the case. Therefore, two parameters $c_x,\ c_y$ are introduced

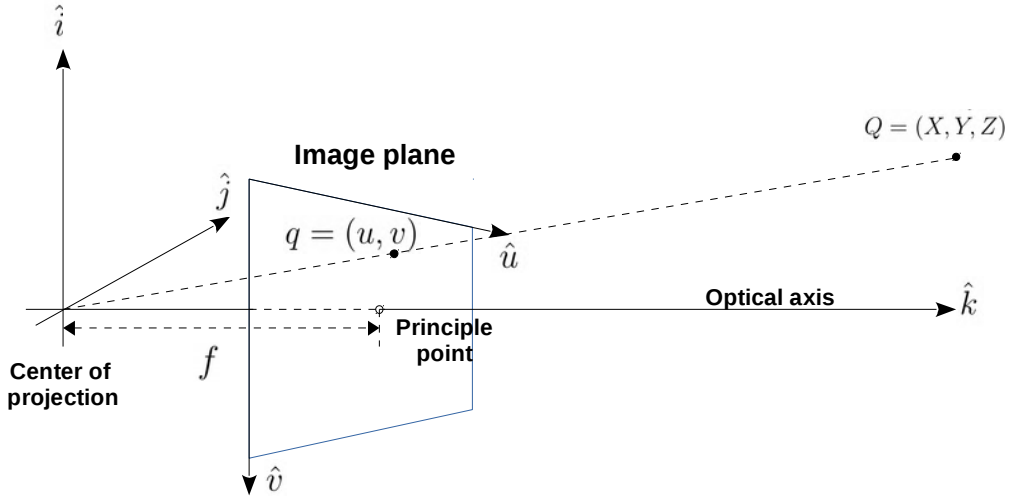


Figure A.2: The pinhole camera model with equivalent and simpler math. A point $Q = (X, Y, Z)$ in the 3D space is projected onto the image plane.

to denote the displacement between the principal point and the image center. Also ideally, there should be only one common focal length f, but due to non-squared pixel, two focal lengths f_x and f_y are introduced. The parameters c_x, c_y and f_x, f_y are the **intrinsic parameters** of this camera model.

Every optical ray starts from a 3D world point and reaches the center of projection. The intersecting point between this ray and the image plane is the 2D projection of the 3D point. As shown in Figure A.2, the image projection coordinates of the 3D point $Q = (X, Y, Z)$ is calculated with

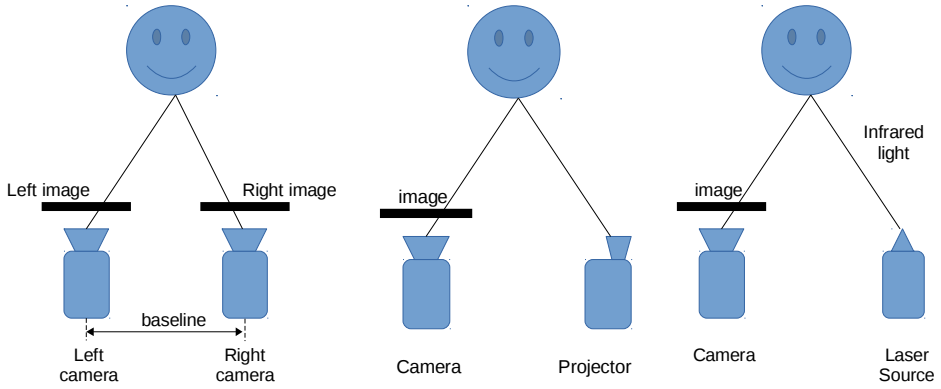
$$u = f_x \frac{X}{Z} + c_x, \quad v = f_y \frac{Y}{Z} + c_y, \tag{A.2}$$

assuming the intrinsic parameters are known.

A.1.2 Depth Map

A depth map is an image or image channel that contains the distance information between a 3D point on a surface, and the image plane. More precisely, it contains the distance information along the $\hat{k}$ axis in Figure A.2. In this case, the projected 2D pixel for Q would contain the value of Z.

There are three commonly used approaches to acquire a depth map, as shown in Figure A.3. The first is through stereo imaging (Figure A.3(a)). The basic idea is emulating human stereo vision with a computational algorithm. Having a pair of cameras perceiving a scene and returning a pair of left and right images, the task of the algorithm is to solve the correspondence problem by finding the same surface points in the left and right image [16]. Then, the depth of those points can be calculated with triangulation. The distance between the two camera centers is called the baseline, and it determines the field of view and the image resolution



(a) Stereo (b) Structured light (c) Time of flight

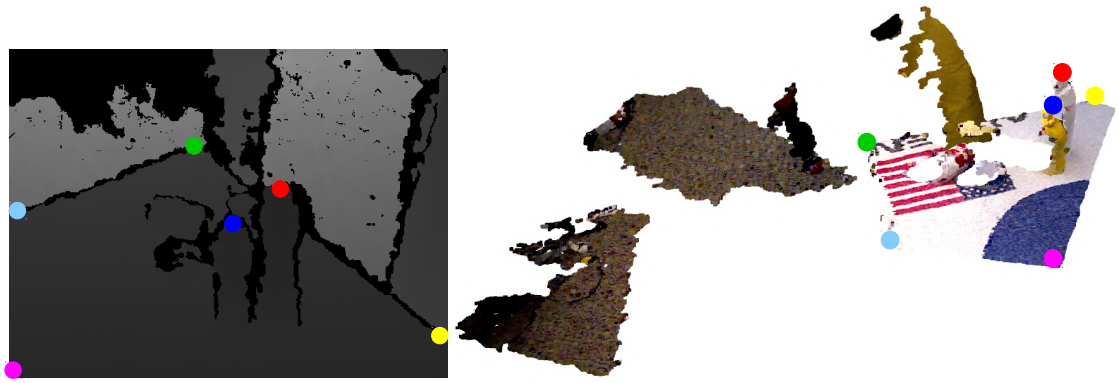
Figure A.3: Methods for acquiring depth maps.

(a) RGB image (b) Depth image

Figure A.4: A pair of RGB (left) and depth images (right).

of the system. The structured light system has a camera and a projector (Figure A.3(b)). The projector projects a known pattern with infrared light onto the scene. The camera captures the deformation of the pattern on surfaces and the depth information is calculated with triangulation. A time of flight system contains a laser light source which functions as the emitter, and a camera sensor which functions as the receiver (Figure A.3(c)). The laser source emits a light beam and the camera sensor measures the time offset, when the beam is reflected back from the object surface. In this way, the depth is measured directly.

An example of depth map is shown in Figure A.4(b). The distance of each pixel is colored coded with gray scale values. The closer the object point is to the image plane, the darker it is. Note that the black pixels denotes missing depth values, and it is normal to observe "holes" in a depth image. The left image (Figure A.4(a)) is its corresponding color image. By appending the depth map to the color image, a 4-channel RGB-D image is obtained.



(a) Depth image (b) Point cloud

Figure A.5: A depth image (left) and its corresponding point cloud with color (right). The colored dots denote the corresponding positions of image points and point cloud points.

A.1.3 Point Cloud

Although a depth map contains the distance "Z" information for point Q (Figure A.2), it is desired to have the full (X, Y, Z) information for a complete 3D perception. Referring to Figure A.2, this problem can be formulated with q and Q. Assuming q is a pixel on a depth map with image coordinates (u, v), its value is Z, which is the depth of 3D point Q, according to equation A.2, X and Y can be obtained with

$$X = \frac{Z}{f_x}(u - c_x), \quad Y = \frac{Z}{f_y}(v - c_y), \tag{A.3}$$

with known camera intrinsics.

Figure A.5 shows an illustration of a depth image and the corresponding point cloud. The colored dots denote the corresponding positions of image points and point cloud points. The 3D information of each point in the point cloud is calculated with the depth map and the camera intrinsic parameters, and the color information is from the RGB color image.

A.2 The Lie Group $SO(3)$

This section provide descriptions on the Lie group and Lie algebra, and how they are used for conversions among different rotation representations.

A.2.1 Lie Group and Lie Algebra

A 3D rotation is a linear transformation in $\mathbb{R}^3$, and it can be represented with a real matrix. All rotation matrices form the group of $SO(3)$, and it is defined [43]

$$SO(3) = \{R \in \mathbb{R}^{3\times3} \mid RR^T = \mathrm{I}_{3\times3}, \det R = 1\}. \tag{A.4}$$

$SO(3)$ is a compact Lie group which contains the 3-by-3 orthogonal matrices with determinant +1.

The Lie group $SO(3)$ is associated with the Lie algebra $so(3)$, which is the set of skew-symmetric 3-by-3 matrices. Those skew-symmetric matrices are the elements of the tangent space of $SO(3)$. Moreover, those matrices are viewed as infinitesimal rotations, which are used to form any rotations in the group (also referred as generators in literature) [34]. The basis of $so(3)$ are three matrices [26]

$$G_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \; G_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \; G_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{A.5}$$

An element of $so(3)$ can be represented as a linear combination of those matrices:

$$w_1 G_1 + w_2 G_2 + w_3 G_3 \in so(3) \tag{A.6}$$

where $w_1, w_2, w_3 \in \mathbb{R}$.

A.2.2 Exponential Map and Logarithm Map

There are two important functions associated to $SO(3)$ and $so(3)$, namely the exponential map and the logarithm map. Those two operations map the element back and forth between $so(3)$ and $SO(3)$:

$$\exp : so(3) \mapsto SO(3) \tag{A.7}$$

$$\ln : SO(3) \mapsto so(3) \tag{A.8}$$

where exp denotes the exponential map and ln denotes the logarithm map.

We simplify the element in equation A.6 with $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}^T \in \mathbb{R}^3$. Its corresponding skew-symmetric matrix $\mathbf{w}_\times$ is a linear combination of the generators

$$\mathbf{w}_\times = w_1 G_1 + w_2 G_2 + w_3 G_3 = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix}. \tag{A.9}$$

The exponential map is the matrix exponential of $\mathbf{w}_\times$

$$\exp(\mathbf{w}_\times) = \mathrm{I}_{3\times 3} + \frac{\sin\theta}{\theta}\mathbf{w}_\times + \frac{1-\cos\theta}{\theta^2}\mathbf{w}_\times^2, \tag{A.10}$$

where $\theta^2 = \mathbf{w}^T\mathbf{w}$ and $\theta = \|\mathbf{w}\|_2$.

Assuming $R \in SO(3)$ and $R = \exp(\mathbf{w}_\times)$, the exponential map can be inverted with logarithm map [26]

$$\theta = \arccos\left(\frac{\mathrm{trace}(R) - 1}{2}\right) \tag{A.11}$$

$$\ln(R) = \frac{\theta}{2\sin(\theta}(R - R^T) \tag{A.12}$$

where $\mathbf{w}$ is the off-diagonal elements of $\ln(R)$. Figure A.6 shows an illustration of the relationship between elements in $SO(3)$ and $so(3)$ spaces.

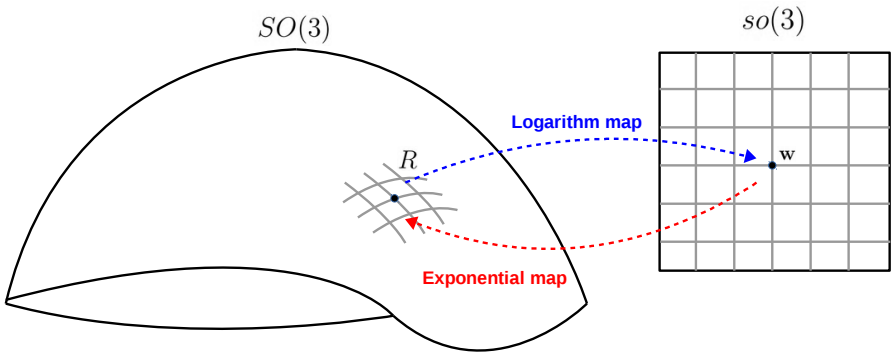


Figure A.6: Illustration of the relationship between elements in $SO(3)$ and $so(3)$ spaces. From an element R in $SO(3)$ represented by a manifold, the logarithm map is used to find its correspondence $\mathbf{w}$ in $so(3)$. And the exponential map is the mapping from $so(3)$ back to $SO(3)$.

A.2.3 Rotation Representation Conversion

The conversion between rotation matrix and axis-angle representation can be done with the exponential map and the logarithm map. Let $\mathbf{r} = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}^T \in \mathbb{R}^3$ be an axis-angle representation of a rotation. It is in the Lie algebra $so(3)$ and the exponential map can be used to obtain the corresponding rotation matrix. Equation A.9 and A.10 can be used to obtain its corresponding 3×3 rotation matrix.

Now let R be a rotation matrix in the Lie group $SO(3)$, and the logarithm map can be used to map it to $so(3)$. Equation A.11 and A.12 can be used to obtain its corresponding axis-angle representation. Assuming $\mathbf{r}$ is the off-diagonal elements of $\ln(R)$, the rotation angle is $\|\mathbf{r}\|$ and the rotation axis is unit vector $\frac{\mathbf{r}}{\|\mathbf{r}\|_2}$.

Given a quaternion $q = q_0 + q_1\hat{\mathbf{i}} + q_2\hat{\mathbf{j}} + q_3\hat{\mathbf{k}}$, its corresponding rotation axis $\mathbf{a}$ and rotation angle θ are

$$\mathbf{a} = (a_x, a_y, a_z) = \frac{(q_1, q_2, q_3)}{\sqrt{q_1^2 + q_2^2 + q_3^2}}, \tag{A.13}$$

$$\theta = 2\text{atan2}(\sqrt{q_1^2 + q_2^2 + q_3^2}, q_0). \tag{A.14}$$

The corresponding rotation matrix R is

$$R = \begin{bmatrix} 1 - 2s(q_2^2 + q_3^2) & 2s(q_1q_2 - q_3q_0) & 2s(q_1q_3 + q_2q_0) \\ 2s(q_1q_2 + q_3q_0) & 1 - 2s(q_1^2 + q_3^2) & 2s(q_2q_3 + q_1q_0) \\ 2s(q_1q_3 - q_2q_0) & 2s(q_2q_3 + q_1q_0) & 1 - 2s(q_1^2 + q_2^2) \end{bmatrix} \tag{A.15}$$

where $s = \|\mathbf{q}\|^2$.

Appendix B

Nomenclature

Abbreviations

3D	3 degrees of freedom
6D, 6-DoF	6 degrees of freedom
ADD	Average distance
ADD-S	Average distance with symmetry
AAE	Augmented autoencoder
AUC	Area under the error threshold-accuracy curve
CAE	Convolutional autoencoder
CNN	Convolutional neural network
DAE	Denoising autoencoder
EV	Explained variation
FPS	Farthest point sampling
HPR	Hidden point removal
ICP	Iterative closest point
KDE	Kernel density estimate
LM	LineMOD
LMO	Occluded LineMOD
MLP	Multi-layer perceptrons
NN	Nearest neighbor
PBR	Physically-based rendering
PPF	Point pair features
REC	Boundary recall

RGB	Red, green and blue
RGB-D	Red, green, blue and depth
SSD	Single shot detector
SVD	Singular Value Decomposition
SIFT	Scale-invariant feature transform descriptor
UE	Undersegmentation error
VCCS	Voxel Cloud Connectivity Segmentation
VGA	Video graphics array
YCBV	YCB video dataset

Frequently Used Symbols

P^O	Object model in object coordinate
P^C	Object segment in camera coordinate
R	3D rotation
$\mathbf{t}$	3D translation

Bibliography

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 34(11):2274–2282, 2012.

[2] Nitin Agarwal, Sung-eui Yoon, and M Gopi. Learning embedding of 3D models with quadric loss. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2019.

[3] K. Somani Arun, Thomas S. Huang, and Steven D. Blostein. Least-squares fitting of two 3D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 9(5):698–700, 1987.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 39(12):2481–2495, 2017.

[5] Sven Bambach, Stefan Lee, David J. Crandall, and Chen Yu. Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1949–1957, 2015.

[6] Annalisa Barla, Francesca Odone, and Alessandro Verri. Histogram intersection kernel for image classification. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages III–513, 2003.

[7] Mark Barnard, Wenwu Wang, and Josef Kittler. Audio head pose estimation using the direct to reverberant speech ratio. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8056–8060, 2013.

[8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 404–417, 2006.

[9] Paul J. Besl and Ramesh C. Jain. Three-dimensional object recognition. *ACM Computing Surveys (CSUR)*, 17(1):75–145, 1985.

[10] Paul J. Besl and Neil D. McKay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239–256, 1992.

[11] Andreea Birhala, Catalin Nicolae Ristea, Anamaria Radoi, and Liviu Cristian Dutu. Temporal aggregation of audio-visual modalities for emotion recognition. In *International Conference on Telecommunications and Signal Processing (TSP)*, pages 305–308, 2020.

[12] Abdeslam Boularias, James Bagnell, and Anthony Stentz. Learning to manipulate unknown objects in clutter by reinforcement. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1336–1342, 2015.

[13] Eric Brachmann. *Learning to predict dense correspondences for 6D pose estimation*. PhD thesis, Technische Universität Dresden, 9 2017. `https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-236564`.

[14] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6D object pose estimation using 3D object coordinates. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 536–551, 2014.

[15] Eric Brachmann, Frank Michel, Alexander Krull, Michael Ying Yang, Stefan Gumhold, and Carsten Rother. Uncertainty-driven 6D pose estimation of objects and scenes from a single RGB image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3364–3372, 2016.

[16] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[17] Mai Bui, Sergey Zakharov, Shadi Albarqouni, Slobodan Ilic, and Nassir Navab. When regression meets manifold learning for object recognition and pose estimation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 6140–6146, 2018.

[18] Roberto Calandra, Andrew Owens, Dinesh Jayaraman, Justin Lin, Wenzhen Yuan, Jitendra Malik, Edward H Adelson, and Sergey Levine. More than a feeling: Learning to grasp and regrasp using vision and touch. *IEEE Robotics and Automation Letters*, 3(4):3300–3307, 2018.

[19] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Benchmarking in manipulation research using the Yale-CMU-Berkeley object and model set. *Robotics & Automation Magazine, IEEE*, 22(3):36–52, 2015.

[20] Alvaro Collet, Manuel Martinez, and Siddhartha S Srinivasa. The MOPED framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research (IJRR)*, 30(10):1284–1306, 2011.

[21] Enric Corona, Kaustav Kundu, and Sanja Fidler. Pose estimation for objects with rotational symmetry. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7215–7222, 2018.

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[23] Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi, and Harinandan Katam. BlenderProc. *arXiv preprint arXiv:1911.01911*, 2019.

[24] Andreas Doumanoglou, Rigas Kouskouridas, Sotiris Malassiotis, and Tae-Kyun Kim. Recovering 6D object pose and predicting next-best-view in the crowd. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3583–3592, 2016.

[25] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3D object recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 998–1005, 2000.

[26] Ethan Eade. Lie groups for 2D and 3D transformations. 2013. `http://ethaneade.com/lie.pdf,revisedDec`.

[27] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing (TIP)*, 6(9):1305–1315, 1997.

[28] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3D object reconstruction from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 605–613, 2017.

[29] Gabriele Fanelli, Matthias Dantone, Juergen Gall, Andrea Fossati, and Luc Van Gool. Random forests for real time 3d face analysis. *International journal of computer vision*, 101(3):437–458, 2013.

[30] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision (IJCV)*, 59(2):167–181, 2004.

[31] David A. Forsyth and Jean Ponce. *Computer vision: a modern approach*. Pearson, 2012.

[32] Simone Frintrop, Thomas Werner, and German Martín García. Traditional saliency reloaded: A good old model in new shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 82–90, 2015.

[33] Huazhu Fu, Xiaochun Cao, Dai Tang, Yahong Han, and Dong Xu. Regularity preserved superpixels and supervoxels. *IEEE Transactions on Multimedia*, 16(4):1165–1175, 2014.

[34] Jean Gallier. *Basics of Classical Lie Groups: The Exponential Map, Lie Groups, and Lie Algebras*, pages 367–414. Springer New York, 2001.

[35] Ge Gao, Mikko Lauri, Xiaolin Hu, Jianwei Zhang, and Simone Frintrop. CloudAAE: Learning 6D object pose regression with on-line data synthesis on point clouds. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021, accepted.

[36] Ge Gao, Mikko Lauri, Yulong Wang, Xiaolin Hu, Jianwei Zhang, and Simone Frintrop. 6D object pose regression via supervised learning on point clouds. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3643–3649, 2020.

[37] Ge Gao, Mikko Lauri, Jianwei Zhang, and Simone Frintrop. Saliency-guided adaptive seeding for supervoxel segmentation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4938–4943, 2017.

[38] Ge Gao, Mikko Lauri, Jianwei Zhang, and Simone Frintrop. Occlusion resistant object rotation regression from point cloud segments. In *Proceedings of the European Conference on Computer Vision Workshops (ECCVW)*, 2018.

[39] Germán M García, Ekaterina Potapova, Thomas Werner, Michael Zillich, Markus Vincze, and Simone Frintrop. Saliency-based object discovery on RGB-D data with a late-fusion approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1866–1873, 2015.

[40] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[42] Frederik Hagelskjær and Anders Glent Buch. PointVoteNet: Accurate object detection and 6 DoF pose estimation in point clouds. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 2641–2645, 2020.

[43] Brian Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*. Springer, 2015.

[44] William Rowan Hamilton. On quaternions, or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 29(191):26–31, 1846.

[45] Shreyas Hampali, Mahdi Rad, Markus Oberweger, and Vincent Lepetit. HOnnotate: A method for 3D annotation of hand and object poses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3196–3206, 2020.

[46] Allan Hanbury. How do superpixels affect image segmentation? In *Proceedings of the Iberoamerican Congress on Pattern Recognition*, pages 178–186, 2008.

[47] Chris Harris and Carl Stennett. RAPID - a video rate object tracker. In *Proceedings of the British Machine Vision Conference*, pages 73–77, 1990.

[48] Richard Hartley, Jochen Trumpf, Yuchao Dai, and Hongdong Li. Rotation averaging. *International journal of computer vision (IJCV)*, 103(3):267–305, 2013.

[49] Yisheng He, Wei Sun, Haibin Huang, Jianran Liu, Haoqiang Fan, and Jian Sun. PVN3D: A deep point-wise 3D keypoints voting network for 6DoF pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11632–11641, 2020.

[50] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 34(5):876–888, 2011.

[51] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 858–865, 2011.

[52] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 548–562, 2012.

[53] Stefan Hinterstoisser, Vincent Lepetit, Naresh Rajkumar, and Kurt Konolige. Going further with point pair features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 834–848, 2016.

[54] Long Hoang, Suk-Hwan Lee, Oh-Heum Kwon, and Ki-Ryong Kwon. A deep learning method for 3D object classification using the wave kernel signature and a center point of the 3D-triangle mesh. *Electronics*, 8(10):1196–1209, 2019.

[55] Tomáš Hodan, Pavel Haluza, Štepán Obdržálek, Jiri Matas, Manolis Lourakis, and Xenophon Zabulis. T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888, 2017.

[56] Tomáš Hodaň, Jiří Matas, and Štěpán Obdržálek. On evaluation of 6D object pose estimation. In *Proceedings of the European Conference on Computer Vision Workshops (ECCVW)*, pages 606–619, 2016.

[57] Tomáš Hodaň, Frank Michel, Eric Brachmann, Wadim Kehl, Anders Glent Buch, Dirk Kraft, Bertram Drost, Joel Vidal, Stephan Ihrke, Xenophon Zabulis, Caner Sahin, Fabian Manhardt, Federico Tombari, Tae-Kyun Kim, Jiří Matas, and Carsten Rother. BOP: Benchmark for 6D object pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[58] Tomáš Hodaň, Martin Sundermeyer, Bertram Drost, Yann Labbé, Eric Brachmann, Frank Michel, Carsten Rother, and Jiří Matas. BOP challenge 2020 on 6D object localization. In *Proceedings of the European Conference on Computer Vision Workshops (ECCVW)*, pages 577–594, 2020.

[59] Tomáš Hodaň, Vibhav Vineet, Ran Gal, Emanuel Shalev, Jon Hanzelka, Treb Connell, Pedro Urbina, Sudipta Sinha, and Brian Guenter. Photorealistic image synthesis for object instance detection. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 66–70, 2019.

[60] Berthold Horn, Berthold Klaus, and Paul Horn. *Robot vision*. MIT press, 1986.

[61] Jan Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. What makes for effective detection proposals? *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 38(4):814–830, 2015.

[62] Du Q Huynh. Metrics for 3D rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

[63] Omid Hosseini Jafari, Siva Karthik Mustikovela, Karl Pertsch, Eric Brachmann, and Carsten Rother. iPose: Instance-aware 6D pose estimation of partly occluded objects. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 477–492, 2018.

[64] Varun Jampani, Deqing Sun, Ming-Yu Liu, Ming-Hsuan Yang, and Jan Kautz. Superpixel sampling networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 352–368, 2018.

[65] Asako Kanezaki and Tatsuya Harada. 3D selective search for obtaining object candidates. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 82–87, 2015.

[66] Sagi Katz, George Leifman, and Ayellet Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21(8):649–658, 2005.

[67] Sagi Katz, Ayellet Tal, and Ronen Basri. Direct visibility of point sets. In *ACM SIGGRAPH*, pages 24–31, 2007.

[68] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1521–1529, 2017.

[69] Wadim Kehl, Fausto Milletari, Federico Tombari, Slobodan Ilic, and Nassir Navab. Deep learning of local RGB-D pacthes for 3D object detection and 6D pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 205–220, 2016.

[70] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5974–5983, 2017.

[71] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: a convolutional network for real-time 6-DOF camera relocalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2938–2946, 2015.

[72] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[73] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[74] Alexander Krull, Eric Brachmann, Frank Michel, Michael Ying Yang, Stefan Gumhold, and Carsten Rother. Learning analysis-by-synthesis for 6D pose estimation in RGB-D images. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 954–962, 2015.

[75] Jack B. Kuipers. *Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality*. Princeton university press, 1999.

[76] Loic Landrieu and Mohamed Boussaha. Point cloud oversegmentation with graph-structured deep metric learning. In *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition, pages 7440–7449, 2019.

[77] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]*, 2, 2010. `http://yann.lecun.com/exdb/mnist`.

[78] Vincent Lepetit. Recent advances in 3D object and hand pose estimation. *arXiv:2006.05927*, 2020.

[79] Alex Levinshtein, Adrian Stere, Kiriakos N Kutulakos, David J Fleet, Sven J Dickinson, and Kaleem Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 31(12):2290–2297, 2009.

[80] Chi Li, Jin Bai, and Gregory D Hager. A unified framework for multi-view multi-class object pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 254–269, 2018.

[81] Gongyang Li, Zhi Liu, and Haibin Ling. ICNet: Information conversion network for RGB-D based salient object detection. *IEEE Transactions on Image Processing (TIP)*, 29:4873–4884, 2020.

[82] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. DeepIM: Deep iterative matching for 6d pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 683–698, 2018.

[83] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 21–37, 2016.

[84] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1150–1157, 1999.

[85] Yan Lu, Yue Wu, Bin Liu, Tianzhu Zhang, Baopu Li, Qi Chu, and Nenghai Yu. Cross-modality person re-identification with shared-specific feature transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13376–13386, 2020.

[86] Vaïa Machairas, Matthieu Faessel, David Cárdenas-Peña, Théodore Chabardes, Thomas Walter, and Etienne Decencière. Waterpixels. *IEEE Transactions on Image Processing (TIP)*, 24(11):3707–3716, 2015.

[87] Fabian Manhardt, Wadim Kehl, and Adrien Gaidon. ROI-10D: monocular lifting of 2D detection to 6D pose and metric shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2069–2078, 2019.

[88] Pat Marion, Peter R Florence, Lucas Manuelli, and Russ Tedrake. Label Fusion: A pipeline for generating ground truth labels for real rgbd data of cluttered scenes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3325–3242, 2018.

[89] David R Martin, Charless C Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(5):530–549, 2004.

[90] Frank Michel, Alexander Kirillov, Eric Brachmann, Alexander Krull, Stefan Gumhold, Bogdan Savchynskyy, and Carsten Rother. Global hypothesis generation for 6D object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 462–471, 2017.

[91] Chaitanya Mitash, Kostas E Bekris, and Abdeslam Boularias. A self-supervised learning system for object detection using physics simulation and multi-view pose estimation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 545–551, 2017.

[92] Alastair P Moore, Simon JD Prince, Jonathan Warrell, Umar Mohammed, and Graham Jones. Superpixel lattices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[93] Hans Moravec. *Mind Children: The Future of Robot and Human Intelligence.* Harvard University Press, 1988.

[94] Markus Oberweger, Mahdi Rad, and Vincent Lepetit. Making deep heatmaps robust to partial occlusions for 3D object pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 119–134, 2018.

[95] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407, 2011.

[96] Seymour A. Papert. The summer vision project. 1966. `http://dspace.mit.edu/handle/1721.1/6125`.

[97] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Wörgötter. Voxel cloud connectivity segmentation - supervoxels for point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2027–2034, 2013.

[98] Kiru Park, Timothy Patten, and Markus Vincze. Pix2Pose: Pixel-wise coordinate regression of objects for 6D pose estimation. In *Proceedings of the*

IEEE International Conference on Computer Vision (ICCV), pages 7668–7677, 2019.

[99] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. PVNet: Pixel-wise voting network for 6DoF pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4561–4570, 2019.

[100] Giorgia Pitteri, Michaël Ramamonjisoa, Slobodan Ilic, and Vincent Lepetit. On object symmetries and 6D pose estimation from images. In *Proceedings of the International Conference on 3D Vision (3DV)*, pages 614–622, 2019.

[101] Charles Ruizhongtai Qi. *Deep learning on point clouds for 3D scene understanding*. PhD thesis, Stanford University, 2018. `https://searchworks.stanford.edu/view/12741586`.

[102] Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum PointNets for 3D object detection from RGB-D data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 918–927, 2018.

[103] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017.

[104] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 5099–5108, 2017.

[105] Mahdi Rad and Vincent Lepetit. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 3828–3836, 2017.

[106] Xiaofeng Ren Ren and Jitendra Malik. Learning a classification model for segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 10–17, 2003.

[107] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571, 2011.

[108] Radu Bogdan Rusu. *Semantic 3D object maps for everyday manipulation in human living environments*. PhD thesis, Technische Universität München, 2009. `http://mediatum.ub.tum.de/doc/800632/941254.pdf`.

[109] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3212–3217, 2009.

[110] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3212–3217, 2009.

[111] Caner Sahin, Rigas Kouskouridas, and Tae-Kyun Kim. A learning-based variable size part extraction architecture for 6D object pose recovery in depth images. *Image and Vision Computing*, 63:38–50, 2017.

[112] Qi She, Fan Feng, Xinyue Hao, Qihan Yang, Chuanlin Lan, Vincenzo Lomonaco, Xuesong Shi, Zhengwei Wang, Yao Guo, Yimin Zhang, Fei Qiao, and Rosa H. M. Chan. OpenLORIS-Object: A robotic vision dataset and benchmark for lifelong deep learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4767–4773, 2020.

[113] Shankar T. Shivappa, Bhaskar D. Rao, and Mohan M. Trivedi. Role of head pose estimation in speech acquisition from distant microphones. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3557–3560, 2009.

[114] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 746–760, 2012.

[115] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. SUN RGB-D: A RGB-D scene understanding benchmark suite. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 567–576, 2015.

[116] Simon Christoph Stein, Florentin Wörgötter, Markus Schoeler, Jeremie Papon, and Tomas Kulvicius. Convexity based object partitioning for robot applications. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3213–3220, 2014.

[117] David Stutz, Alexander Hermans, and Bastian Leibe. Superpixels: An evaluation of the state-of-the-art. *Computer Vision and Image Understanding (CVIU)*, 166:1–27, 2018.

[118] Hao Su, Charles Ruizhongtai Qi, Yangyan Li, and Leonidas J. Guibas. Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2686–2694, 2015.

[119] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. Implicit 3D orientation learning for 6D object detection from RGB images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 699–715, 2018.

[120] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[121] Alykhan Tejani, Danhang Tang, Rigas Kouskouridas, and Tae-Kyun Kim. Latent-class Hough forests for 3D object detection and pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 462–477, 2014.

[122] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. Real-time seamless single shot 6D object pose prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 292–301, 2018.

[123] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *Proceedings of the Conference on Robot Learning (CoRL)*, pages 306–316, 2018.

[124] Roy Uziel, Meitar Ronen, and Oren Freifeld. Bayesian adaptive superpixel segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 8470–8479, 2019.

[125] Michael Van den Bergh, Xavier Boix, Gemma Roig, Benjamin de Capitani, and Luc Van Gool. SEEDS: superpixels extracted via energy-driven sampling. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 13–26, 2012.

[126] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[127] Andrea Vedaldi and Stefano Soatto. Quick shift and kernel methods for mode seeking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 705–718, 2008.

[128] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12):3371–3408, 2010.

[129] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 511–518, 2001.

[130] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. DenseFusion: 6D object pose estimation by iterative dense fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3343–3352, 2019.

[131] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J. Guibas. Normalized object coordinate space for category-level 6D object pose and size estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2642–2651, 2019.

[132] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019.

[133] David Weikersdorfer, Alexander Schick, and Daniel Cremers. Depth-adaptive supervoxels for RGB-D video segmentation. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 2708–2712, 2013.

[134] Yilin Wen, Hao Pan, Lei Yang, and Wenping Wang. Edge enhanced implicit orientation learning with geometric prior for 6D pose estimation. *IEEE Robotics and Automation Letters (RAL)*, 5(3):4931–4938, 2020.

[135] Christian Wilms and Simone Frintrop. Superpixel-based refinement for object proposal generation. In *International Conference on Pattern Recognition (ICPR)*, pages 1–8, 2020.

[136] Daniel Wolf, Johann Prankl, and Markus Vincze. Enhancing semantic segmentation for robotics: the power of 3D entangled forests. *IEEE Robotics and Automation Letters (RAL)*, 1(1):49–56, 2016.

[137] Jian Wu, Liwei Ma, and Xiaolin Hu. Delving deeper into convolutional neural networks for camera relocalization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5644–5651, 2017.

[138] Jimmy Wu, Bolei Zhou, Rebecca Russell, Vincent Kee, Syler Wagner, Mitchell Hebert, Antonio Torralba, and David M.S. Johnson. Real-time object pose estimation with pose interpreter networks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6798–6805, 2018.

[139] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: a convolutional neural network for 6D object pose estimation in cluttered scenes. In *Proceedings of the Robotics: Science and Systems (RSS)*, 2018.

[140] Yanyang Xiao, Zhonggui Chen, Zhengtao Lin, Juan Cao, Yongjie Jessica Zhang, Yangbin Lin, and Cheng Wang. Merge-swap optimization framework for supervoxel generation from three-dimensional point clouds. *Remote Sensing*, 12(3):473–497, 2020.

[141] Chenliang Xu and Jason J Corso. Evaluation of super-voxel methods for early video processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1202–1209, 2012.

[142] Fengting Yang, Qian Sun, Hailin Jin, and Zihan Zhou. Superpixel segmentation with fully convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13964–13973, 2020.

[143] J. Yang, Z. Gan, K. Li, and C. Hou. Graph-based segmentation for RGB-D data using 3-D geometry enhanced superpixels. *IEEE Transactions on Cybernetics*, 45(5):927–940, 2015.

[144] Jian Yao, Marko Boben, Sanja Fidler, and Raquel Urtasun. Real-time coarse-to-fine topologically preserving segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2947–2955, 2015.

[145] Dongbo Yu, Jun Xiao, and Ying Wang. High-precision plane detection method for rock-mass point clouds based on supervoxel. *Sensors*, 20(15):4209–4231, 2020.

[146] Shanxin Yuan, Qi Ye, Bjorn Stenger, Siddhant Jain, and Tae-Kyun Kim. Bighand2.2m benchmark: Hand pose dataset and state of the art analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4866–4874, 2017.

[147] Christopher Zach, Adrian Penate-Sanchez, and Minh-Tri Pham. A dynamic programming approach for fast and robust object pose recognition from range images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 196–203, 2015.

[148] Sergey Zakharov, Wadim Kehl, Benjamin Planche, Andreas Hutter, and Slobodan Ilic. 3D object instance recognition and pose estimation using triplet loss with dynamic margin. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 552–559, 2017.

[149] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. DPOD: 6D pose object detector and refiner. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1941–1950, 2019.

[150] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the

ADE20K dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.

[151] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: a modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[152] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5745–5753, 2019.

[153] Ming Zhu, Chao Ma, Pan Ji, and Xiaokang Yang. Cross-modality 3D object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3772–3781, 2021.

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum Unterschrift

Erklärung der Urheberschaft

Hiermit versichere ich an Eides statt, dass ich die vorliegende Dissertation im Studiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

Erklärung zur Veröffentlichung

Ich stimme der Einstellung der Dissertation in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum Unterschrift